A Templating System to Generate Provenance

Luc Moreau, Belfrit Victor Batlajery, Trung Dong Huynh, Danius Michaelides, Heather Packer

Abstract—PROV-TEMPLATE is a declarative approach that enables designers and programmers to design and generate provenance compatible with the PROV standard of the World Wide Web Consortium. Designers specify the topology of the provenance to be generated by composing templates, which are provenance graphs containing variables, acting as placeholders for values. Programmers write programs that log values and package them up in sets of bindings, a data structure associating variables and values. An expansion algorithm generates instantiated provenance from templates and sets of bindings in any of the serialisation formats supported by PROV. A quantitative evaluation shows that sets of bindings have a size that is typically 40% of that of expanded provenance templates and that the expansion algorithm is suitably tractable, operating in fractions of milliseconds for the type of templates surveyed in the article. Furthermore, the approach shows four significant software engineering benefits: separation of responsibilities, provenance maintenance, potential runtime checks and static analysis, and provenance consumption. The article gathers quantitative data and qualitative benefits descriptions from four different applications making use of PROV-TEMPLATE. The system is implemented and released in the open-source library ProvToolbox for provenance processing.

Index Terms—provenance, PROV, provenance generation, template

1 Introduction

PROVENANCE has gained a lot of traction lately in various areas including the Web, legal notices[1], climate science[2], scientific workflows [1], [2], [3], computational reproducibility [4], emergency response [5], medical applications[3], geospatial domain[4], art and food. The recent standard PROV [6] of the World Wide Web Consortium defines provenance as "as a record that describes the people, institutions, entities, and activities involved in producing, influencing, or delivering a piece of data or a thing." In an increasing number of applications, provenance has become crucial in making systems accountable, by exposing how information flows in systems, and in helping users decide whether information is to be trusted. Provenance is not restricted to computer systems, it can also be used to describe how objects are transformed and people are involved in a physical system [5].

Applications and use cases for provenance are well documented in the literature [7], [8], [9], [10]. They include making systems more auditable and accountable [11], reproducing results [12], deriving trust and classification [13], asserting attribution and generating acknowledgments [14], supporting predictive analytics [13], and facilitating traceability [15]. To enable such a powerful functionality, however, one needs to adapt or write applications, so that they generate provenance information, which can then be exploited to offer new benefits to their users.

A number of approaches have been proposed to generate provenance: run-time, compile-time, and retrospectively. Runtime generation typically requires applications to be instrumented, and provenance generated accordingly [16], [17], [18]. Instrumenting applications and generating provenance at the same time can be cumbersome from a software engineering perspective. Instead, traditional logging techniques have been combined with provenance generation [19]. Workflow systems are a class of applications generating provenance, which use a mix of instrumentation and logging (cf. Taverna [1], Vistrails [2], Kepler [3]). Aspect-oriented approaches have also been used to weave provenance generation instructions into programs [20]. In contrast, compile-time generation uses static analysis, such as dependency analysis [21] and program slicing [22], to produce executables that generate provenance information. In many situations, however, we have to deal with legacy applications, for which we do not have the opportunity to modify the source code to introduce provenance-generating code, or we do not have the opportunity to recompile programs. Thus, in those cases, using application knowledge, provenance can alternatively be *reconstructed* retrospectively [23], [24].

Regardless of the approach, whether by instrumentation, logging, aspects, static methods, or reconstruction, at some point, a provenance record needs to be constructed. This may involve writing code that generates a provenance record in a well-defined, standard serialization format, such as RDF [25], text [26] or XML [27]. Alternatively, a toolkit can be used to create a memory representation of provenance and to serialize it (e.g. ProvPy [28] and ProvToolbox [29]). Writing the code for generating provenance and its serialization is error-prone since, to be inter-operable [30], it is required to address all the idiosyncrasies of the model and formats. Whilst using libraries facilitates this task, the programming effort may have to be repeated across the whole of the application's code base. This is particularly challenging since, provenance being still a rather new concept, programmers are generally not familiar with its technical details, and are not the best placed to fine-tune the provenance information to be generated. This problem is particularly compounded in large projects with code

• *The authors are with the Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, Southampton, UK.*
E-mail: l.moreau@ecs.soton.ac.uk

Manuscript revised January 23, 2017

1. https://www.thegazette.co.uk/
2. http://nca2014.globalchange.gov/report
3. https://www.hl7.org/fhir/provenance.html
4. http://www.opengeospatial.org/projects/initiatives/ows-10

developers distributed geographically or across different organisations: specifically, when agile methods are being adopted, the programming overhead makes it very difficult to maintain consistent and timely updates to the provenance generation code, so that the provenance it generates remains aligned with the actual behaviour of the application.

To address these challenges, we propose PROV-TEMPLATE, a templating approach for generating PROV-compliant provenance, with the following original and distinct characteristics:

1) With PROV-TEMPLATE, a designer can express the shape of the provenance to be generated; the provenance templates offer a declarative specification of provenance, rather than stating how it has to be generated.
2) Provenance templates are simply provenance documents expressed in a PROV-compatible format and containing placeholders, referred to as "variables", for values.
3) A mechanism allows for values to be injected into those placeholders: PROV-TEMPLATE is equipped with an expansion algorithm that, given a template and a set of bindings (associating variables to values), generates a provenance record in one of the standardized PROV representations.

Given that we have argued against crafting code to generate provenance, we felt strongly that it was not suitable to define yet another specialized language to generate provenance. Hence, the originality of PROV-TEMPLATE is that provenance templates are expressed in PROV directly. A simple serialization format also exists for sets of bindings. These design constraints have been adopted with a view to minimize the number of specific notations designers and developers have to learn.

From a software development viewpoint, we have observed a number of benefits of the PROV-TEMPLATE approach:

1) Separation of Responsibilities. Provenance template design and maintenance become the responsibility of "knowledge engineers," whereas provenance-related coding is reduced to bindings generation that can be embraced by a distributed development team, without PROV expertise.
2) Maintenance. It becomes possible to maintain an application-wide library of templates in a single location, allowing for incremental updates of templates, for instance, due to changes to application ontology definitions.
3) Runtime or Static Checks. A number of safety checks can be introduced such as determining whether bindings are compatible with templates; this includes type-compatibility, arity, mandatory/optional nature, potential semantic constraints, and the ability to check whether templates are valid [31].
4) Provenance Consumption. An application consuming the provenance it generates can directly process sets of bindings, rather than perform graph queries, followed by some query result post-processing.

We initially released a specification [32] of PROV-TEMPLATE and an implementation in ProvToolbox [29] in 2014. Building on our experience of using the system in various projects, the aim of this article is to present PROV-TEMPLATE, with the following original contributions: i) an overview of the approach and its positioning in the software development cycle; ii) a description of the template language, the sets of bindings, the expansion algorithm, and core checks; iii) a quantitative evaluation of the template expansion approach, showing improved performance when manipulating bindings over expanded provenance templates; iv) a qualitative discussion of our practical experience with PROV-TEMPLATE.

The article is organized as follows. Section 2 consists of an example of template as well as a brief introduction to PROV, and the challenges associated with the programmatic generation of PROV. Section 3 then presents the architectural overview of PROV-TEMPLATE. Section 4 defines the templates, bindings and expansion algorithm. The quantitative evaluation is then the focus of Section 5, whereas our practical experience with PROV-TEMPLATE is discussed in Section 7. We compare our approach to related work in Section 8, before concluding the article in Section 9.

2 Provenance Applications and Example

As a way of providing motivations for provenance, we outline four applications exploiting provenance (Section 5 contains an evaluation that is based on templates and bindings from these applications). One of the applications, EBook (see Section 2.1.3), serves as an illustration to convey the intuition of PROV and the templating approach.

2.1 Four Provenance-Enabled Applications

2.1.1 Smartshare

Smartshare[5] [33] is a "car pooling" application that allows drivers and commuters to offer and request rides. Ride offers and requests include details about required travels, timing, locations, capacity, prices, and other details relevant to car sharing. The application automatically matches commuters to available cars. It is fully provenance-enabled, capturing the provenance of any user decision, matching or rating managed by the system. The purpose of provenance in Smartshare is to make the application accountable, in particular, by providing explanations about all decisions made. The smart data set in this article consists of all the provenance bindings and associated templates involved in six ride plans leading to two agreed rides between two users.

2.1.2 Food

The food application tracks food orders and deliveries in Hampshire schools in England, with a view to develop "due diligence" methods with scientific authorities of the county. The purpose of provenance in this application is to describe the origin of food and develop analytics methods over the food supply chain. The food data set in this article consists of the templates that describe orders, deliveries, food specifications and sampling, and associated bindings for six schools in the county over six months.

5. http://www.smart-society-project.eu/software/smartshare/

2.1.3 EBook

The EBook project[6] released a suite of tools designed to aid in the use and teaching of reproducible statistical analysis techniques with a particular emphasis on their use in social science. It consists of a workflow system capable of logging provenance, and a system to convert provenance back into workflows. The purpose of provenance in EBook is to support the aim of reproducibility of scientific experiments [12]. The ebook data set in this article consists of the single template used by the EBook system and sets of bindings generated by the execution of a specific statistical workflow.

2.1.4 PICASO

The PICASO application[7] (Provenance Interlinking and Collective Authoring for Scientific Objects) is an online platform that crowdsources the links between related scientific objects identified by Unified Resource Identifiers (URI). In PICASO, the purpose of provenance is to form a knowledge graph linking all scientific work; it is published as linked open data to allow for further analyses and research over this kind of information. The picaso data set in this article consists of all the templates supported by PICASO and bindings for some 4000 entries.

2.2 An Example of Template

In this section, we present an example of template and sets of bindings and provide a brief overview of the PROV data model. For further and normative details, we refer the reader to the PROV specifications [6], [25], [26].

The template depicted in Figure 1 is a simplification of the template used in the EBook application (see Section 2.1.3). It captures the following provenance pattern: an agent has launched a workflow, consisting of execution steps, referred to as blocks [34], each consuming some input files, and generating output files. An input or output file is modelled as a `prov:Entity` (represented by yellow ellipses in Figure 1, labeled as "consumed" and "produced", respectively); the execution of the workflow or a workflow step is modelled as a `prov:Activity` (represented as blue rectangles, labeled as "parent" and "block_instance", respectively); finally, the user is modelled as a `prov:Agent` (represented as an orange pentagon labeled "agent"). A few relations interconnect these nodes: the output file is derived from the input file (relation `prov:wasDerivedFrom`), the input is used by the block (relation `prov:used`), whereas the output is generated by it (relation `prov:wasGeneratedBy`), the execution step was started by the parent execution (relation `prov:wasStartedBy`), whereas the parent workflow was associated with the user (relation `prov:wasAssociatedWith`). Further attributes of a block include its type, start and end time, etc. We note that PROV relations are expressed using the past tense to highlight that provenance is intended to be a description of something that happened in the past (as opposed to a workflow specification or a program aimed to be executed in the future).

6. http://www.bristol.ac.uk/cmm/research/ebooks/
7. https://provenance.ecs.soton.ac.uk/picaso/

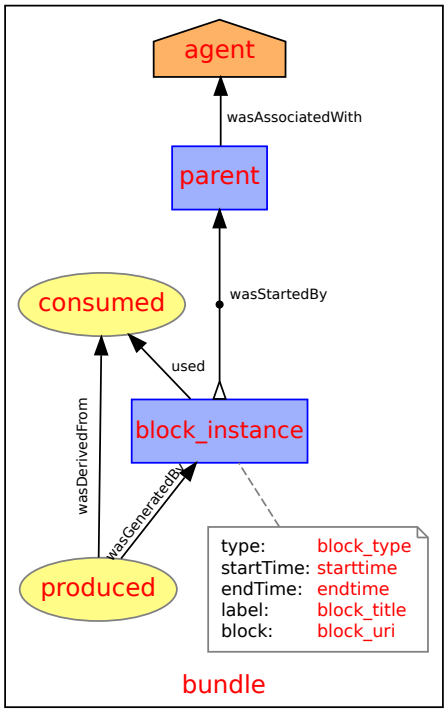

Fig. 1. A graphical illustration of a template. It shows an activity `block_instance`, which used an entity and generated another entity, the latter derived from the former. The activity was started by a `parent` activity (itself a block or an overarching workflow). An agent is associated with the `parent` activity. Some properties are associated with the `block_instance` activity, including its type, its start and end time, a human-readable label and a URI to its full description. Terms appearing in red are variables acting as placeholders for values.

Whilst Figure 1 is purely illustrative, Figure 2 presents the same template using the normative PROV-N notation [26]. Each node and relation of the graph is expressed by a PROV-N statement. The PROV-N notation makes it explicit which name is a placeholder for values: all names in the namespace[8] `var` are regarded as variables by PROV-TEMPLATE (they are marked in red in both Figures 1 and 2). Note that some of the variables are not displayed in Figure 1 to preserve its legibility.

The template of Figure 1 describes the execution of an activity `block_instance`. A typical workflow consists of multiple steps, and the execution of each of these steps has to be described by instantiating the same template, with bindings pertaining to this specific execution.

In Figure 1 the graph is displayed inside a box labeled by variable `bundle`; likewise, in Figure 2, we can see that PROV terms occur inside a `bundle` construct, with the same variable. A bundle is a PROV construct [6] that allows provenance of provenance to be expressed. Specifically, such bundles are present in templates so that their attribution can be expressed, allowing the process of template expansion to be documentable by provenance itself. Although such an attribution is permitted, it is not included in the example for

8. Figure 2 also contains names in the namespace `t` of PROV-TEMPLATE. We refer the reader to the ProvToolbox template user's guide [32] for details of how such names are used to control the expansion of variables meant to generate time and string values. Such low-level practical details are not discussed any further in this paper.

```
document
 prefix t <http://openprovenance.org/tmpl#>
 prefix var <http://openprovenance.org/var#>
 prefix estat <http://purl.org/net/statjr/ns#>
 prefix estatwf <http://purl.org/net/statjr/wf#>

 bundle var:bundle
  activity(var:block_instance,
           [ t:startTime='var:starttime',
             t:endTime='var:endtime',
             prov:type='var:block_type',
             t:label='var:block_title',
             estatwf:block='var:block_uri' ])
  activity(var:parent)
  agent(var:agent)
  wasAttributedTo(var:parent,var:agent)
  wasStartedBy(var:block_instance, -, var:parent,
               -, [t:time='var:starttime'])
  entity(var:consumed)
  used(var:block_instance, var:consumed, -,
       [ t:time='var:consumed_at',
         estat:bindingname='var:consumed_name'])
  entity(var:produced)
  wasGeneratedBy(var:produced, var:block_instance,
       -, [ t:time='var:produced_at',
            estat:bindingname='var:produced_name'])
  wasDerivedFrom(var:produced, var:consumed)
 endBundle
endDocument
```

Fig. 2. Template of Figure 1 expressed in PROV-N. Variables are qualified names with prefix `var` and appear in red.

clarity. For instance, one could express that the template is part of a template library of an application; this also allows details of the template expansion to be captured (such as the version of the library that performed the expansion at a specific date).

2.3 Bindings and Template Expansion

An extract from a set of bindings is displayed in Figure 3: it is a JSON structure that contains a dictionary for the various variables (in red), mapping them to one or more values (in blue). For instance, there are two consumed inputs, so there are two values for the variable "consumed", expressed as UUIDs denoting each of the two inputs; on the other hand, there is a single output, so there is a single value for the variable "produced".

The expansion of the template of Figure 2 with the bindings of Figure 3 is illustrated graphically in Figure 4, and its PROV-N representation is shown in Figure 5. Variables have been replaced by values (displayed in blue in Figure 5). We see that two entities are consumed (`fe1bf93c-...` and `0266d11a-...`). Supplementary Material contains a fuller example of template expansion.

2.4 The Difficulty of Generating Provenance Without Template

The four applications (Smartshare, Food, EBook, PICASO) introduced in Section 2.1 use templates to generate provenance. Prior to designing PROV-TEMPLATE, we developed applications that were creating provenance directly: software engineers had to design, implement and maintain code that generated provenance similar to that of Figure 5. This presented a number of challenges, which we illustrate, based on our concrete experience with several applications.

```
{"var": {
  "consumed": [
    { "@id": "urn_uuid:fe1bf93c- ..." },
    { "@id": "urn_uuid:0266d11a- ..." } ],
  "consumed_at": [
    { "@type": "xsd:dateTime",
      "@value": "2016-03-08T14:21:12.085706" },
    { "@type": "xsd:dateTime",
      "@value": "2016-03-08T14:21:12.085706" } ],
  "produced": [
    { "@id": "urn_uuid:0266d372- ..." } ],
  "produced_at": [
    { "@type": "xsd:dateTime",
      "@value": "2016-03-08T14:21:12.085819" } ],
  "block_instance": [
    { "@id": "urn_uuid:0266c6c0- ..." } ],
  "endtime": [
    { "@type": "xsd:dateTime",
      "@value": "2016-03-08T14:21:12.085844" } ],
  "block_type": [
    { "@id": "estatwf:BuiltinFunction" }  ],
  "agent": [
    { "@id": "estatwf:John" }  ],
  "parent": [
    { "@id": "urn_uuid:0266c558- ..." }  ],
   ...  },
 "context": {
     "xsd": "http://www.w3.org/2001/XMLSchema#",
     "estatwf": "http://purl.org/net/statjr/wf#",
     "urn_uuid": "urn:uuid:" } }
```

Fig. 3. An example of set of bindings for the template of Figure 2. A set of bindings is encoded as a dictionalry associating variables to values. Variable names are shown in red, and values in blue.

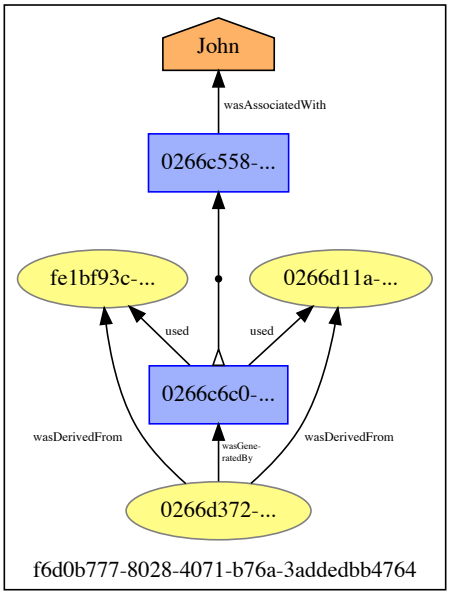

Fig. 4. A graphical illustration of the expansion of template of Figure 1 with bindings of Figure 3.

ProvToolbox [29] is an open-source general-purpose toolkit to manipulate PROV-based provenance in Java. It comes with a few examples of Java programs to create a memory representation of provenance and save it to various PROV serialization formats. One of the examples is the Provenance Challenge workflow [35], a reference workflow for the provenance community. An implementation can be

```
document
 bundle uuid:f6d0b777-8028-4071-b76a-3addedbb4764
  prefix uuid <urn:uuid:>
  prefix estatwf <http://purl.org/net/statjr/wf#>
  prefix uuid <urn:uuid:>
   used(uuid:0266c6c0-...,uuid:fe1bf93c-...,
        2016-03-08T14:21:12.085Z)
   used(uuid:0266c6c0-...,uuid:0266d11a-...,
        2016-03-08T14:21:12.085Z)
   activity(uuid:0266c6c0-...,
            2016-03-08T14:21:12.085Z,
            2016-03-08T14:21:12.085Z)
   activity(uuid:0266c558-...,-,-)
   entity(uuid:0266d11a-...)
   entity(uuid:fe1bf93c-...)
   wasAssociatedWith(uuid:0266c558-...,
                     estatwf:John,-)
   agent(estatwf:John)
   entity(uuid:0266d372-...)
   wasDerivedFrom(uuid:0266d372-...,
                  uuid:fe1bf93c-...)
   wasDerivedFrom(uuid:0266d372-...,
                  uuid:0266d11a-...)
   wasStartedBy(uuid:0266c6c0-...,-,
                uuid:0266c558-...,
                2016-03-08T14:21:12.085Z)
   wasGeneratedBy(uuid:0266d372-...,
                  uuid:0266c6c0-...,
                  2016-03-08T14:21:12.085Z)
 endBundle
endDocument
```

Fig. 5. The expanded provenance of Figure 4 expressed in PROV-N. Following expansion, variables have been replaced by values shown in blue

found on Github[9] and is also discussed in more details in Supplementary Material. To a first approximation, each PROV term (such as those occurring in Figure 5) requires a factory method to be called; potentially, a further method call would be required to add each key-value pair occurring in these terms. As a graph structure is being built, each node needs to be identified by a URI or a qualified name (i.e., a URI short form [26]), and likewise, each relation connects two or more nodes, also identified by some URIs. This essentially requires code that manipulates names with a size proportional to that of the provenance graph to be generated. This results in a significant burden on software developers since they would be required to understand the semantics of these constructor methods, and would have to ensure that the graph is constructed correctly, associating resources of types compatible with the edge semantics, and making sure that edges are constructed with the right directions. As an illustration, 200 lines of Java code are required to compose the Provenance Challenge provenance graph, which can be broken down into 50 lines for node constructors, 130 for edge constructors, and 15 for adding attributes. As some nodes have up to 8 incoming edges, there is a significant amount of repetition in node names provided to edge constructors, and therefore of opportunities to code the graph construction incorrectly.

ProvToolbox [29] and ProvPy [28] are libraries that take care of deserializing to, and serializing from, Java and Python representations, respectively. As an illustration, Table 1 provides an approximate line count of this functionality. A developer intending to generate directly a serialization of PROV, would be confronted with developing and maintaining a code base whose size is a significant fraction of the ones illustrated in Table 1.

TABLE 1
Number of lines of code to support various serialization formats of PROV in ProvToolbox and ProvPy. The difference in size can be attributed to (i) Java's verbosity and (ii) programming styles favouring dynamic typing and reflection in Python and static typing in Java.

format	ProvToolbox lines	ProvPy lines
PROV-XML	13000 (incl. beans)	330
rdf	3600	550
json	1600	310
PROV-N	1600	100 (output only)

The serialization formats come with their idiosyncrasies, and as specifications are revised, or when incorrect implementations have to be fixed, non-trivial work is required. The type of effort to maintain such libraries is illustrated by the following change to ProvPy. Previously, ProvPy used `prov:QualifiedName` (implemented by the `QualifiedName` class) to denote short forms of URIs, and also supported `xsd:QName` backed by `XSDName` class, a subclass of `QualifiedName`. However, to ensure better compliance with PROV-N, support for `xsd:QName` was removed, and instead the new `prov:QUALIFIED_NAME` with changed capitalization had to be supported. Changes affected some 60 lines [10]. In Section 7.3, we revisit this change and show how it affect the PICASO application that is built on ProvPy and templates.

CollabMap is a provenance-enabled crowdsourcing application [13]. Provenance is used in CollabMap for auditing the application's behavior and for predicting the quality of data produced by the crowd [13]. CollabMap was being developed as the standardization of PROV began, and therefore, relies on a predecessor model of provenance called OPM, the Open Provenance Model; it also predates ProvPy. A significant part of CollabMap code specifically aims at provenance management: about 145 lines are about constructing provenance (out of 400 related to the data model), and some 80 lines are concerned with JSON export, and 50 for RDF export (out of 700 of the view part of the MVC model). Handcrafting the PROV-JSON structure was tedious and error-prone, and changes required code edits. Issues that had to be handled included: lack of default namespace required to resolve qualified names without a prefix, referring to attribute with a string representation instead of a qualified name, upgrading to the PROV model as its specification was being agreed.

In summary, the set of evidence we have presented point towards the need for a principled method for generating provenance and associated tooling, such as PROV-TEMPLATE, which we now describe in details.

9. Provenance Challenge implemented with ProvToolbox is available at https://github.com/lucmoreau/ProvToolbox/tree/development/tutorial/tutorial5.

10. See the changes at https://github.com/trungdong/prov/commit/c7e21a9cbc551187cb335b7fa28032ef79d695c8.

3 Architectural Overview

Figure 6 provides an overview of the PROV-TEMPLATE architecture. In blue, we show key facets of runtime execution: an application logs values, in the form of bindings, which are used by the template expander to generate provenance documents. The template expander relies on a template for provenance created at design-time; the template may refer to application or domain ontologies. In red, we show the templates and ontologies which are created at design time.

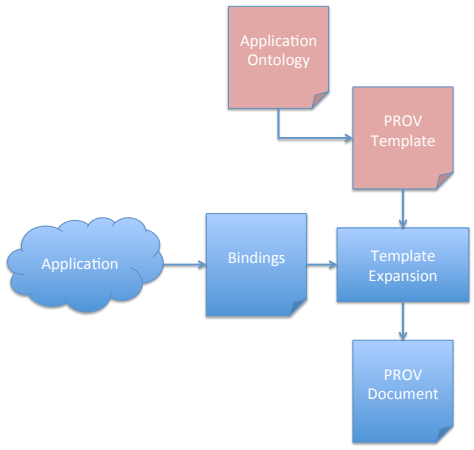

Fig. 6. The architecture of the templating approach. The blue and red colors are used to refer to runtime and design time aspects of the approach, respectively.

The architecture is agnostic about the mechanism used to create bindings. Applications may log values, and then convert them to bindings. Alternatively, *aspects* may be weaved into the code to generate such bindings. The architecture is also agnostic about when provenance is generated and which documents are persisted. Template expansions may be interleaved with application execution, possibly persisting provenance graphs; alternatively, bindings may be accumulated, and provenance generated for post-execution analysis. The specific requirements of the application with respect to provenance determines when these operations have to take place.

4 Template, Binding, Template Expansion

In this section, we present a conceptualization of PROV-TEMPLATE, including the abstract syntax of templates, a definition of sets of bindings, and a description of the expansion algorithm.

4.1 Template Definition

Figure 7 displays the abstract syntax of templates. In this article, the presentation is concerned with the syntactic dimension of the PROV data model; for the detailed and normative meaning of these constructs, we refer the reader to the PROV data model specification [6].

A template, denoted by the meta-variable $template$ is a named set of terms (referred to as "bundle" [6]). Terms can be of five kinds and are denoted by the meta-variables $term_i$ for $i = 1, \ldots, 5$. Nodes, denoted by $term_1$, can be entities, activities, or agents; they are identified by a mandatory name μ. The remaining terms are relations, connecting such nodes. In the simplest case, relations denoted by $term_2$ are binary associations, connecting two nodes denoted by their names α and β. Qualified relations [25], expressed by meta-variable $term_3$, enrich binary relations with a set of attribute-value pairs and an optional identifier τ. Relations described by $term_4$ and $term_5$ further include one or more secondary names δ, allowing qualified relations to be refined with extra information. When attribute-value pairs are contained in a term, they occur in a set, which allows a given attribute key k_i to be present multiple times with different values; their order is not significant.

Constants ($\pi \in \mathcal{P}$) can be of usual primitive types (for instance, XML Schema built-in datatypes [36]); constants consist of two elements: their external representation as a string, and their type. For example, when we conveniently write `"John Doe"`, we mean the constant with external representation "`John Doe`" and type xsd:string. Likewise, the conveniently written number `40`, in fact, denotes the constant with external representation "`40`" and type xsd:integer.

The grammar of templates also allows for a name γ to appear in attribute-value position, associated with an attribute key κ. In that case, there is an implicit conversion of the name γ into a constant in $\mathcal{P}$, consisting of the external representation of γ and the reserved type prov:QUALIFIED_NAME. In other words, all constants with this reserved type are considered as names by PROV-TEMPLATE. Such constants are abbreviated with the PROV-N single quote notation [26] (for instance, `'var:endtime'` in Figure 2).

Names are said to be qualified [25], consisting of a prefix denoting a namespace URI and a local name. The template language comes with its own namespace, in which some names are defined with a specific meaning.

prefix	namespace URI
t	`http://openprovenance.org/tmpl#`

4.2 Simple Set of Bindings

The set of all names $Names(t)$ in a template t is the set of all names $\alpha, \beta, \delta, \gamma, \mu, \tau$, for all terms in template t. We distinguish two categories of names, according to their positions in terms.

In the definition of a template in Figure 7, the names $\alpha, \beta, \delta, \mu, \tau$ act as placeholders — or variables — for some concrete names in $\mathcal{N}$. The process of template expansion replaces such placeholders by concrete names. The expansion algorithm expects a template and a set of bindings, which maps names to their concrete instantiation.

In contrast, in Figure 7, an attribute variable name γ acts as a placeholder for some constants. Let us consider a binding that maps a name γ to a set of constants $\pi_0, \pi_1, \ldots$. Then, the expansion of an attribute-value pair $\kappa = \gamma$ results in a series of attribute-value pairs $\kappa = \pi_0, \kappa = \pi_1, \ldots$.

We can formally define sets of bindings as follows.

Definition 1 (Simple set of bindings). A simple set of bindings ρ for a term $term_t$ (resp. a template t) is a

		Terms	Indexing Names $I(.)$	PROV Predicates
$template$	::=	bundle$(\mu, term^*)$	μ	
$term$	::=	$term_1 \mid term_2 \mid term_3 \mid term_4 \mid term_5$		
$term_1$	::=	$node(\mu, [\kappa_i = \gamma, \ldots, \kappa_j = \pi, \ldots])$	μ	$node \in$ {ent, act, ag}
$term_2$	::=	$rel_1(\beta, \alpha)$	α, β	$rel_1 \in$ {spec, alt, mem}
$term_3$	::=	$rel_2(\tau; \beta, \alpha, [\kappa_i = \gamma, \ldots, \kappa_j = \pi, \ldots])$	α, β	$rel_2 \in$ {wgb, used, wat, winvb, winfb, winflb}
$term_4$	::=	$rel_3(\tau; \beta, \alpha, \delta, [\kappa_i = \gamma, \ldots, \kappa_j = \pi, \ldots])$	α, β, δ	$rel_3 \in$ {assoc, del, wsb, web}
$term_5$	::=	$rel_4(\tau; \beta, \alpha, \delta_1, \delta_2, \delta_3, [\kappa_i = \gamma, \ldots, \kappa_j = \pi, \ldots])$	$\alpha, \beta, \delta_1, \delta_2, \delta_3$	$rel_4 \in$ {der}

Sets:

$$\begin{aligned} \mathcal{N} &= \{\eta_0, \eta_1, \ldots\} \text{ Set of names, with total order } \eta_i < \eta_j \text{ if } i < j \\ \mathcal{P} &= \{\pi_0, \pi_1, \ldots\} \text{ Set of constants} \\ \mathcal{K} &= \{\kappa_0, \kappa_1, \ldots\} \text{ Set of keys} \end{aligned}$$

A name position within a term t determines its kind:

	Kind of Names		Kind of Names
τ, δ, γ	optional name	α	influencer
μ, α, β	mandatory name	β	influencee
μ, τ	identifying name	γ	attribute value

Fig. 7. Abstract syntax of the template language.

partial map of names of the term $Names(term_t)$ (resp. the template $Names(t)$) to some value, whether it is a name in $\mathcal{N}$ or a set of constants in $\mathcal{P}$.

It is useful for subsequent formalizations to consider a set of bindings as a total map. Given a template t and a simple set of bindings ρ, a total, simple set of bindings ρ_T for t is a total function mapping each name of the template $Names(t)$ to some value, whether it is a name or a set of constants. For any $\nu \in Names(t)$,

$$\rho_T(\nu) = \begin{cases} \rho(\nu) & \text{if } \rho(\nu) \text{ is defined} \\ \nu & \text{if } \rho(\nu) \text{ is not defined.} \end{cases}$$

4.3 Simple Name Replacing in Templates

Given a simple set of bindings ρ, we can replace a template's names by the values associated with them in ρ. However, there are three ways of doing this, which we refer to as *permissive*, *strict*, and PROV-*aware* name replacing, respectively denoted by $replace_{perm}$, $replace_{strict}$, and $replace_{pa}$. These three ways of replacing names correspond to different modes a developer may want to use the PROV-TEMPLATE system. Permissive mode, for instance, allows partial instantiation of templates. Strict mode is particularly useful when debugging to ensure that all variables have been instantiated. Finally, PROV-aware mode is an intermediate way of operating, taking into account the optional nature of some names. We now define them in turn.

Permissive name replacing is the function that substitutes the names found in a set of bindings ρ, and leaves the others untouched. For instance, permissive name replacing of $term_3$, written $replace_{perm}(term_3, \rho)$, is defined as the term $term'_3$:

$$term'_3 = rel_3(\tau'; \beta', \alpha', \delta', [\kappa_i = \gamma', \ldots, \kappa_j = \pi, \ldots])$$

where each name ν' is obtained by $\nu' = \rho_T(\nu)$, where ν ranges over $\alpha, \beta, \delta, \nu, \tau$. For a pair $k_i = \gamma$ in $term_3$, a name γ occurring in attribute-value position is allowed to be bound to multiple values. If γ is unbound, or bound to a single name, then we find an attribute key-value pair $\kappa_i = \gamma'$ in $term'_3$, with $\gamma' = \rho_T(\gamma)$. If γ is bound to a set of constants, for each $\pi \in \rho_T(\gamma)$, there is an attribute key-value pair $\kappa_i = \pi$ in $term'_3$. We note that ρ_T is used, ensuring that names unbound in ρ are kept unchanged (since ρ_T maps them to themselves).

Strict name replacing requires all names to be bound by the set of bindings. For instance, strict name replacing of $term_3$, written $replace_{strict}(term_3, \rho)$, is defined as the same term $term'_3$, under the condition that $\rho(\nu)$ is defined for each name ν ranging over $\alpha, \beta, \delta, \gamma, \mu, \tau$. If $\rho(\nu)$ is not defined for one of the names ν, then the strict replacing operation is not defined for the whole term.

However, neither permissive nor strict replacing suitably takes the PROV semantics into account. The names α and β are mandatory, whereas names τ, δ and names in attribute-value position γ are all optional. Thus, we introduce the *provenance-aware replacing strategy*, $replace_{pa}$. If there is no binding for optional names, they are replaced by the distinguished symbol $-$ for τ, δ; likewise, an attribute-value pair $\kappa = \gamma$ is not included, if there is no binding for name γ. If a binding is missing for a mandatory name, then the whole replacing operation fails, like in the strict strategy. So, let us define

$$\rho_-(\nu) = \begin{cases} \rho(\nu) & \text{if } \rho(\nu) \text{ is defined} \\ - & \text{if } \rho(\nu) \text{ is not defined.} \end{cases}$$

We define $replace_{pa}(term_3, \rho) = term'_3$, if $\alpha' = \rho(\alpha), \beta' = \rho(\beta)$ are both defined, and $\tau' = \rho_-(\tau), \delta' = \rho_-(\delta)$. For a pair $\kappa_i = \gamma$ in $term_3$, if γ is bound to a single name, there is an attribute key-value pair $\kappa_i = \gamma'$, with $\gamma' = \rho(\gamma)$. If γ is bound to a set of constants, for each $\pi \in \rho(\gamma)$, there is an attribute key-value pair $\kappa_i = \pi$. If $\rho(\gamma)$ is not defined, then the pair is not included in $term'_3$.

For $term'_3$ to remain syntactically correct, each name in ρ must be bound to a name, except for those names occurring only in attribute-value position (γ), which are allowed to be bound to any constant.

Of course, in the special case when a name ν occurs in attribute-value position, such as in $\kappa = \nu$, and also elsewhere in a term, if $\rho(\nu) = \nu'$, then an implicit conversion takes place into a set of constants $\{\nu'\}$ allowing instantiation into the attribute-value pairs $\kappa = \nu$.

4.4 Linked Names

Let us imagine that we have to design the provenance for a book; attribution could be used to link the book to its author. In the case of multiple authors, we do not want to have to specify a template for each possible number of authors. Instead, we prefer to define a template containing one attribution relation between a book and an author, and provide bindings for multiple authors.

For simple binary relations, denoted by $term_2$ in Figure 7, it may be interesting to specify their type (one-to-one, one-to-many, many-to-one, many-to-many), so that when a name is given multiple values, it is easy to understand how to expand a template. However, $term_3$, $term_4$, and $term_5$ show that we are not just considering binary relations, but generalized n-ary relations over names. So the question is: if names are to be bound to multiple values, which names are expected to be given a similar number of values, and how should the expansion proceed. Against this background, PROV-TEMPLATE introduces a notion of linked name.

Definition 2 (Linked Name). Two names ν_1, ν_2 in a template t are said to be linked if $Linked_t(\nu_1, \nu_2)$ holds, where the relation $Linked_t$ is obtained by the symmetric, transitive closure of the relation $Link$, iteratively computed as follows for all terms of t:

1) If $node(\mu, [\text{t:linked} = \gamma, \ldots])$, then $Link(\mu, \gamma)$.
2) If $rel_i(\tau; \beta, \alpha, \ldots, [\text{t:linked} = \gamma, \ldots])$ for $i \in \{2, 3, 4\}$, then $Link(\tau, \gamma)$.

By extension, $Names(Linked_t)$ denotes the set of names related by $Linked_t$.

Example 1. Let us consider the template of Figure 2. There is no occurence of the t:linked attribute. Therefore, $Linked_t = \emptyset$ and $Names(Linked_t) = \emptyset$.

Example 2. Let us call "linked-Figure 2" a variant of template of Figure 2, in which the variables `var:consumed` and `var:produced` are linked, meaning that for each output there is a single input.

```
entity(var:consumed,
       [t:linked='var:produced'])
```

In that case, $Linked_t$ = ((`var:consumed`, `var:produced`), (`var:produced`, `var:consumed`)) and $Names(Linked_t)$ = {`var:produced`, `var:consumed`}.

Given the $Linked_t$ relation, we can construct a partition of names. Furthermore, given a strict ordering of names, we can construct a unique sequence of names sets, as follows. The following definition relies on the notion of indexing name, which can be found for each term in Figure 7.

Definition 3 (Link-Partition). Let $I(term_t)$ be the set of indexing names in a term $term_t$ from a template t. Let $P(term_t)$ be the set of partitionable names, defined as the union of $I(term_t)$ and $\{\gamma \mid \gamma \in term_t \wedge \gamma \in Names(Linked_t)\}$. A link-partition is a sequence $N_0, N_1, \ldots, N_{m-1}$ of sets of names such that $\forall k, 0 \leq k < m, \forall i, j, 0 \leq i < j < m$, the following holds:

1) $\cup_i N_i = P(term_t)$;
2) $N_i \cap N_j = \emptyset$;
3) $\forall \nu_m, \nu_n \in N_k, Linked_t(\nu_m, \nu_n)$;
4) $\exists \nu^i \in N_i$, such that $\forall \nu^j \in N_j, \nu^i < \nu^j$.

In Definition 3, Clauses 1 and 2 show that $N_0, N_1, \ldots, N_{m-1}$ is a partition. Clause 3 states that each N_k contains linked names. Clause 4 relies on name ordering over $\mathcal{N}$ introduced in Figure 7. We are now equipped with a mechanism that allows us to uniquely enumerate names according to the way they have been linked.

Example 3. Let us consider term

```
wasDerivedFrom(var:produced, var:consumed)
```

in Figure 2. $I(.)$ for this term is {`var:produced`, `var:consumed`}. So is $P(.)$. Its link-partition is N_0 = {`var:consumed`}, and N_1 = {`var:produced`}, assuming that we used lexicographic order over variable names.

Example 4. Let us consider term

```
wasGeneratedBy(var:produced,
               var:block_instance,
             [ t:time='var:produced_at',
               estat:bindingname
                 ='var:produced_name'])
```

in template "linked-Figure 2" described in Example 2. $I(.)$ for this term is {`var:produced`, `var:block_instance`}. So is $P(.)$. Its link-partition is N_0 = {`var:block_instance`}, and N_1 = {`var:produced`}.

Example 5. Instead, if we consider term

```
wasDerivedFrom(var:produced,
               var:consumed),
```

in template "linked-Figure 2" described in Example 2. $I(.)$ for this term is {`var:produced`, `var:consumed`}. So is $P(.)$. Its link-partition is N_0 = {`var:produced`, `var:consumed`}.

4.5 Complex Sets of Bindings and Template Expansion

While a simple set of bindings allows for one value (either a name or a set of constants) for each variable, a complex set of bindings allows for multiple values for variables. Such bindings enable cross-products of values to be created by template expansion. Thus, a complex set of bindings for a template is defined as a total function ϕ_t that maps each name of the template to a vector of names (from $\mathcal{N}$) or vectors of sets of values (from $\mathcal{P}$).

$\phi_t : Names(t) \rightarrow Vector(\mathcal{N}) \cup Vector(\mathbb{P}(\mathcal{P}))$

A Link-Partition indicates which partitionable names are linked. When multiple bindings are supported, names in a given partition are regarded as an array of names, being simultaneously assigned values identified in bindings. Therefore, the number of values associated with names in a complex set of bindings must be the same, if those names belong to the same partition; this number provides

the number of possible assignments for the names of the partition. This is formalized as follows.

Definition 4 (Binding-Partition Compatibility 1). Let us consider $P(term_t)$, the set of partitionable names in a term $term_t$ from template t; its link-partition $N_0, N_1, \ldots, N_{m-1}$, and a complex set of bindings ϕ_t. A complex set of bindings ϕ_t is compatible with the link-partition $N_0, N_1, \ldots, N_{m-1}$, if the following holds:

$$\forall k, 0 \leq k < m, \forall \nu_i, \nu_j \in N_k, \\ length(\phi_t(\nu_i)) \quad = \quad length(\phi_t(\nu_j)),$$

i.e., the number of possible bindings for two names ν_i, ν_j from the same partition is the same. Let us denote this number as $size^k_{\phi_t}$ for N_k.

Therefore, an integer ι in $[0, size^k_{\phi_t} - 1]$ can act as an index for a value in the vector $\phi_t(\nu)$ for any $\nu \in N_k$.

Example 6. Building on Example 3, the complex set of bindings $\phi_t = \{$`var:produced` $\rightarrow \langle p_1, p_2\rangle$, `var:consumed` $\rightarrow \langle c_1\rangle\}$, which defines two generated entities for one used entity is compatible with the link-partition of Example 3 and thus satisfies Definition 4.

Example 7. Following on Example 4, the same complex set of bindings $\phi_t = \{$`var:produced` $\rightarrow \langle p_1, p_2\rangle$, `var:consumed` $\rightarrow \langle c_1\rangle\}$ is not compatible with Definition 4, because the number of values associated with `var:consumed` and `var:produced` is not the same.

What about names that are neither indexing nor linked? The purpose of Definition 5 is to set expectations for the number of values to be found associated with those names in a complex set of bindings.

Definition 5 (Binding-Partition Compatibility 2). Let us consider $P(term_t)$, the set of partitionable names in a term $term_t$ from template t; its link-partition $N_0, N_1, \ldots, N_{m-1}$, and a complex set of bindings ϕ_t. A complex set of bindings ϕ_t is compatible with the link-partition $N_0, N_1, \ldots, N_{m-1}$, if the following holds:

$$\forall \nu, \nu \notin P(term_t), \\ length(\phi_t(\nu)) \quad = \quad 0 \text{ or } size^0_{\phi_t} \times \ldots size^{m-1}_{\phi_t},$$

i.e., the number of bindings for a non-partitionable name ν is given by the number of possible combinations with all partitionable names.

Example 8. Building again on Example 3, $\phi_t = \{$`var:produced` $\rightarrow \langle p_1, p_2\rangle$, `var:consumed` $\rightarrow \langle c_1\rangle$, `var:block_instance` $\rightarrow \langle b_1\rangle$, `var:produced_at` $\rightarrow \langle t_1, t_2\rangle\}$ satisfies Definition 5, because it has 2 bindings for `var:produced_at`, since $size^0_{\phi_t} = 1$ and $size^1_{\phi_t} = 2$.

Binding-Partition compatibility requires both Definitions 4 and 5 to be satisfied.

Definition 6 (Multi-Index). Let us consider $P(term_t)$, the set of partitionable names in a term $term_t$ from template t; its link-partition $N_0, N_1, \ldots, N_{m-1}$, and a complex set of bindings ϕ_t compatible with the link-partition.

A multi-index $\omega = \langle \iota_0, \iota_1, \ldots, \iota_{m-1}\rangle \in I\!N_0 \times I\!N_1 \times \ldots I\!N_{m-1}$ is a tuple of naturals such that each ι_k belongs to interval $[0, size^k_{\phi_t}[$ for $0 \leq k < m$.
Multi-indices $\omega_0, \omega_1, \ldots$ can be ordered lexicographically. Let us call lex the function that returns the position in this lexical sequence for any multi-index.

Example 9. Using the bindings ϕ_t of Example 8, for which we have link-partition N_0, N_1 of Example 3, we have $\omega_0 = \langle 0, 0\rangle$, and $\omega_1 = \langle 0, 1\rangle$. The lex funtion is defined as $lex(0) = \omega_0$ and $lex(1) = \omega_1$.

A multi-index enables us to extract a simple set of bindings from a complex set of bindings, by means of a projection operation defined as follows.

Definition 7 (Projection). Let us consider $P(term_t)$, the set of partitionable names in a term $term_t$ from template t; its link-partition $N_0, N_1, \ldots, N_{m-1}$, a complex set of bindings ϕ_t compatible with the link-partition, and a multi-index $\omega = \langle \iota_0, \iota_1, \ldots, \iota_{m-1}\rangle$.
The projection operation $project(\phi_t, \omega)$ is a simple set of bindings ρ, such that:

- $\forall k \in [0, m-1], \nu \in N_k, \rho(\nu) = \phi_t(\nu)[\iota_k]$, or
- $\forall \nu \in Names(term_t), \nu \notin P(term_t), \rho(\nu) = \phi_t(\nu)[lex(\omega)]$.

Example 10. Using the multi-index of of Example 9 and the bindings ϕ_t of Example 8, the projection $project(\phi_t, \omega_1)$ is $\{$`var:produced` $\rightarrow \langle p_2\rangle$, `var:consumed` $\rightarrow \langle c_1\rangle$, `var:block_instance` $\rightarrow \langle b_1\rangle$, `var:produced_at` $\rightarrow \langle t_2\rangle\}$, in which the values for the second produced entity have been seelcted.

We are now ready to define template expansion. We start with the expansion of a term.

Definition 8 (Term Expansion). Let us consider $P(term_t)$, the set of partitionable names in a term $term_t$ from template t; its link-partition $N_0, N_1, \ldots, N_{m-1}$, and a complex set of bindings ϕ_t compatible with the link-partition. Let Ω_{term_t} be the set of all multi-indices for term $term_t$.
The expansion of $term_t$ for ϕ_t is the set of terms defined as:
$expansion(term_t, \phi_t) =$

$$\{s_\omega \mid \omega \in \Omega, \\ replace_{pa}(term_t, project(\phi_t, \omega)) \text{ is defined}, \\ s_\omega = replace(term_t, project(\phi_t, \omega))\}.$$

where $replace$ denotes one of the functions $replace_{perm}$, $replace_{strict}, replace_{pa}$.

ProvToolbox [29], which contains an implementation of PROV-TEMPLATE, allows users to choose a replacement function. For instance, the *strict* mode, which terminates with an error code if some template variables have not been bound, is particularly useful when debugging the application since it allows us to check whether bindings are fully constructed. When an application is deployed, we would use the provenance-aware mode, which drops unbound optional variables. In Supplementary Material, we show an application of the permissive approach, in which templates get partially instantiated to create new templates.

Finally, we can define a template expansion as the union of all expansions of terms, as per Definition 8.

Definition 9 (Template Expansion). Given a template $t =$ bundle μ $term^*$ and a complex set of bindings ϕ_t:

$$expansion(t, \phi_t) = \text{bundle } \mu' \{\cup_{term \in term^*} expansion(term, \phi_t)\}$$

with $\mu' = \phi_t(\mu)$.

It is assumed here that a single value is provided for the bundle name.

5 Quantitative Evaluation

In the PROV-TEMPLATE approach, templates specify the topology of the provenance to be generated, whereas bindings contain values required to create the provenance. Given that bindings contain no topological information, the intuition is that bindings are more compact than the expanded templates. The first part of this section contrasts the size of bindings and the size of expanded provenance templates, systematically, across the range of templates supported by the applications discussed in Section 2.1. We observe that the size of bindings is on average 40% of the size of expanded templates. This saving is beneficial in terms of both reduced communication cost and reduced storage cost. Indeed, application components only need to submit bindings to a provenance repository, instead of expanded provenance; a more compact representation of bindings reduces communication overheads. Likewise, one may consider provenance repositories that only persist bindings, instead of expanded provenance: they will result in more compact storage.

The savings in communication and storage should be understood in the context of the extra cost of expanding templates with some bindings. We show that the cost of expansion itself is very modest. On average, across the applications considered, it takes 0.23ms to expand a template, which would allow over 4000 expansions to take place on a single core every second.

For the quantitative evaluation, we consider the four applications presented in Section 2.1, which are referred to as smart (Section 2.1.1), food (Section 2.1.2), ebook (Section 2.1.3), and picaso (Section 2.1.4). Table 2 summarizes the number of bindings sets and templates per application.

TABLE 2
Number of bindings sets and templates per application

application	sets of bindings	templates
smart	1608	12
food	1031	3
ebook	235	1
picaso	4019	13
all	6893	29

The following evaluation relies on ProvToolbox [29], a Java library to manipulate PROV representations, and which includes an implementation of PROV-TEMPLATE. The performance evaluation was run on a MacBook Pro OSX 10.1, with an Intel Core i7, 2.7 GHz, and 16Gb of Memory.

We adopted the following procedure for preparing the test data. PROV has multiple serializations (RDF, XML, Text) but does not have a canonical representation. Thus, to be able to compare sizes of bindings with sizes of expanded provenance templates, we applied the following transformations. Provenance files are converted to Turtle [37], with all qualified names expanded as URIs, and then converted to PROV-N [26], allowing for new namespace prefixes to be automatically allocated. The conversion to Turtle allows terms to be merged [31], where appropriate, whereas the conversion to PROV-N allows for a compact representation. Bindings are serialized to JSON, as in Figure 3 but without pretty printing.

Having prepared the data, we applied the following method to generate Figure 8, which shows the compaction ratio for each template.

1) Apply the expansion algorithm for each set of bindings and template pair.
2) Compute the compaction ratio by dividing the size of each set of bindings (as a JSON file), by the size of the expanded provenance templates (as a PROV-N file, prepared as above).
3) Create the box plot as per Figure 8, where the x-axis enumerates the templates, and the y-axis indicates the compaction ratio.

The x-axis of Figure 8 lists the templates, grouped per application (see the legend for the application color coding). The y-axis indicates the compaction ratio. A compaction ratio equal to 1 means that the size of a set of bindings is the same as the size of an expanded template. The smaller the compaction ratio, the more "efficient" the representation of a set of bindings is. For each template, a box plot shows the median compaction ratio, the first quartile and third quartile, and the minimum and maximum compaction ratios.

Each of the 6893 sets of bindings resulted in a compaction ratio less than one. Table 3 provides a numerical summary of Figure 8 per application. On average, across all the sets of bindings, the compaction ratio is 40%.

TABLE 3
Summary of compaction ratios

application	mean	std. dev.	median
smart	0.430	0.073	0.447
food	0.526	0.080	0.532
ebook	0.670	0.062	0.660
picaso	0.350	0.053	0.342
all	0.406	0.102	0.397

Figure 8 shows some variability in the range of compaction ratios for some templates. For instance, in ebook's `block_run` template, the bindings leading to the smallest ratio (0.35) are about a block with a large number of outputs (31), whereas, at the other end of the scale, bindings with ratio 0.8 are all for blocks that consume and produce one entity. Likewise, the food application's `foodspec` template was designed to contain general descriptions of food specification. The bindings resulting in the lowest ratio (0.24) contain very little details about the food product, whereas the bindings resulting into the highest (0.73) contain long

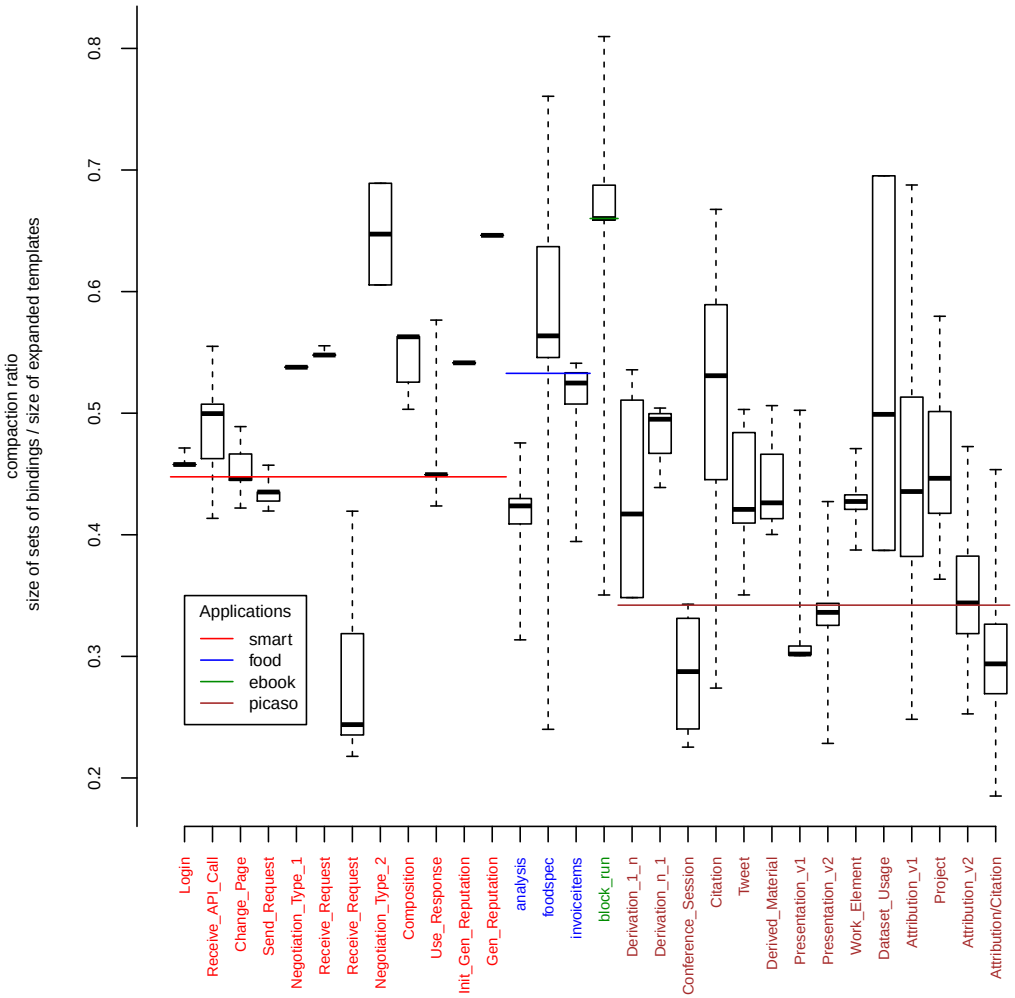

Fig. 8. Compaction ratio, per application, per template. The legend shows the color adopted for each application. Colored horizontal lines are the median compression ratio for each application.

textual strings overwhelming the topology found in the expanded template.

We followed a similar procedure to produce performance data. (A plot is available in Supplementary Material.)

1) For each set of bindings and template pair:
   a) Run the expansion algorithm w times
   b) Repeat c times:
      - Measure average time over n template expansions
   c) Compute average over c measures
   d) Normalize measure with respect to length of the sets of bindings.
2) Create a box plot with templates in the x-axis and normalized averages in the y-axis.

We ran the process with $(w, n, c) = (1000, 40, 20)$. The value w was selected to introduce a delay before taking measurements, to allow for JVM warm up. Of course, the bigger a template is, the longer its expansion; likewise, the bigger a set of bindings is, the longer the expansion is. For the box plot (available in Supplementary Material), to be able to make meaningful comparisons, we normalized the computed average time with respect to the size of sets of bindings in kilobytes.

It is important to note that this experiment only measures expansion time, and does not include the input/output time necessary to read templates and bindings, and write the expanded template.

Table 4 summarizes the raw expansion times (without normalization) for the various applications. We see that in average the expansion time is 0.23ms. This type of performance would allow over 4000 expansions to be performed on a single core per second. This shows that the approach is entirely tractable. Furthermore, it is to be noted that our implementation follows the definitions of Section 4 and currently does not optimize the expansion process. Section 9 suggests ways of improving the performance of template expansion.

We observe that the average expansion time is significantly larger for the food application, but likewise, the

TABLE 4
Summary of expansion time (in ms) — mean, standard deviation, median — and average bindings set size (in bytes)

application	Expansion Time			bindings
	mean	std. dev.	median	size
smart	0.181	0.119	0.147	1106
food	0.605	0.379	0.451	3367
ebook	0.174	0.067	0.160	1126
picaso	0.174	0.100	0.165	875
all	0.234	0.225	0.167	1282

average size of sets of bindings for this application is larger. This confirms our initial hypothesis that the larger a set of bindings, the longer the expansion time. To validate this, we applied Pearson's correlation test and obtained a ρ-value of 0.8879575 and a p-value 2.2×10^{-16} showing a strong correlation.

The sets of bindings in the four applications extensively used the ability to provide multiple values for variables. However, only in a couple of cases did combinatorial explosions occur. Indeed, either the corresponding templates used linked variables to assign values simultaneously, or only one variable was given multiple values, while the others had a single value, resulting in 1-to-n or n-to-1 relations in the expanded template. The most notable case of combinatorial explosion occurred in ebook (template `block_run`) and involved a binary relation with two values for one variable, and 31 for the other, resulting in 62 instances. With the combinatorial effect on the expanded template, the compression ratio is the lowest (0.319), whereas the absolute expansion time is the largest (1.04ms), though its normalized expansion time is not an outlier.

6 Bindings Generation

The quantitative evaluation of Section 5 demonstrated that bindings are a more compressed representation than that of expanded templates. Bindings representation is more space efficient because it is devoid of topological information: definitions of nodes and edges are to be found in template definitions, whereas the bindings only contain associations between variables and values to instantiate them. This efficient representation also brings some benefits in terms of programming the generation of bindings.

First, we discuss some techniques to create bindings easily and efficiently (Section 6.1). Second, we examine how the development environment can help check whether bindings are well-formed and are aligned with template definitions (Section 6.2). In Supplementary Material, all these techniques are illustrated by examples or code fragments.

6.1 Ease and Performance of Generation

We consider four different techniques to generate bindings. Depending on the context of the application, they can be potentially combined together.

6.1.1 Abstract Bindings Creation

As discussed in Section 2.4, programming the generation of a provenance graph typically involves a method call for constructing each node and each edge. This requires the programmer to understand the topology of graphs to be generated. Further, these method calls should receive all necessary values to specify the attributes of these nodes and edges. This also typically involves a substantial amount of repetition in the code, since, for each node, there will be incoming and outgoing edges that require the node identifier (or a reference to it) to be passed to the edge constructors.

Instead of this, a library to construct bindings enables the programmer simply to identify which variable is associated with which value(s). Thus, no requirement is put on the programmer to understand the graph topology and replicate node identifiers across constructors of its adjacent edges. A library can take care of serializing the in-memory representation of bindings to their concrete serial format, relieving the programmer from knowing the bindings syntax. ProvToolbox provides a reference implementation of such abstract bindings.

6.1.2 Concrete Bindings Creation

Given the simple syntax of bindings as a JSON dictionary, it is also easy for the programmer to generate their textual representation directly, or build a dictionary structure (say in Javascript) that serializes directly to JSON. Such technique is particularly useful for programming languages that do not have a library for abstract bindings. It is used in the smart application, in which various components logged bindings constructed in their serialization format.

6.1.3 Converting Tabular Values

Application data is often available or exportable in tabular format, for instance, in the standardized CSV format [38]. If columns are labeled with the names of variables, each row can be converted into a set of bindings, whether abstract (Section 6.1.1) or concrete (Section 6.1.2). Such technique is used in the food application, in which food-related data, already existing in a tabular format, is converted to bindings.

6.1.4 Bindings Fragments

Let us consider the template of Figure 1; an activity may run for a long time. If bindings can only be generated at the end of an activity, it means that there may be portions of provenance that may not become available for a long time. This also places an unnecessary burden on the bindings generation code to hold on to values, until the last one becomes available, potentially resulting in memory leaks. In such a case, it may become desirable to create *bindings fragments*, containing bindings for a subset of the variables of a template. Such technique is used in the ebook application, in which bindings fragments are logged asynchronously, and a separate process reconstructs whole bindings out of fragments extracted from the log. The decoupling of bindings generation and provenance generation is critical to preserving the application's performance.

6.2 Support for Checks

A potential challenge with the above techniques is that, while the programmer's task is facilitated because there is no requirement to program the topology of provenance

graphs, a potential new source of error comes with variables names, and the burden of ensuring that they correspond to the variables occurring in the templates. To address this problem, a constructor of bindings can be generated automatically from a template definition, creating methods such as `addConsumed` in charge of adding a binding for the variable `consumed`. If a variable is renamed in a template, then the bindings constructor can be regenerated. At compile-time, it can be detected if the application code refers to the older method name. ProvToolbox provides an implementation of the bindings generator.

Another type of check that can be performed on a set of bindings is related to the number of values associated with variables occurring in a specific group, as defined in Section 4. The picaso application uses further template metadata to control the user interface, to generate bindings that satisfy constraints on the number of values for variables (for instance, related to minimal or maximal variable cardinality).

7 Practical Experience

In this section, first, we discuss the granularity of templates and bindings; second, we revisit the benefits introduced in Section 1, and provide some evidence supporting these, from the four applications that adopted PROV-TEMPLATE.

7.1 Templates and Bindings Granularity

A question a designer inevitably faces is the granularity with which templates, and to some extent bindings, should be associated with computational modules. We have encountered different cases. In smart, a template is typically associated with a method, or a sequence of method invocations, when it is desirable to abstract away from them. In the food application, the granularity of templates is dictated by the data the application ingests (invoice, food specification, inspection report): templates are designed to be general so that, for instance, they can accommodate invoices from multiple food suppliers. In ebook, a template corresponds to a workflow step. Finally, in picaso, each template corresponds to an editable user interface, visualizing the template, allowing values to be dragged on the interface, to create bindings for that template. In general, if a REST application has to be instrumented to generate provenance, templates could be associated with REST operations.

For smart, food, and picaso, a full set of bindings is created and submitted, all at once, for template expansion. On the other hand, in ebook, as workflow steps can be long, bindings fragment can be submitted independently (See Section 6.1.4).

7.2 Benefit 1: Separation of Responsibilities

The smart application was iteratively developed by four organizations, which contributed various aspects of the design and implementation of three components: user interface (UI), a ride matching service (Orchestrator) and a feedback and rating service (Reputation System). Each component recorded provenance using PROV-TEMPLATE exposed as a web service allowing sets of bindings to be submitted for expansion by the components distributed across the Web. The first stage of development required a PROV expert to design templates in consultation with each component's lead developer, so as to reflect the component's business logic in the template. PROV expertise was required to design the templates in order to support the provenance use cases [39] targeted by the application. In the second phase, the initial integration was completed via pair programming (involving a developer and a PROV expert), to ensure that the correct sets of bindings were submitted for the expected template. Since bindings creation can be error-prone in the absence[11] of automated checks (see Section 6.2), and the component developers were not familiar with PROV and PROV-TEMPLATE details, such a type of pair programming was regarded as the most efficient way of minimizing development effort. Over time, as better tools and better training material become available, the needs for a provenance expert will be significantly reduced (see future work in Section 9). In the third phase, further changes to the values of the variables in the bindings were completed by the components' developers with essentially no support required.

7.3 Benefit 2: Maintenance

Templates may need to be changed as applications are redesigned and evolve, potentially due to new requirements or bug fixes. In turn, bindings may be required to change. In this section, we overview broad categories of changes that may be applied to templates, we then review how bindings generated according to techniques of Section 6 may have to be changed.

Table 5 summarizes broad types of template changes. Like code, templates may need to be refactored: templates may be *renamed* (1), while their contents remain unchanged. A new template may be *added* (2), when a new component is added to an application or a behavior of the application needs to be described by further provenance. On the contrary, components may be decommissioned and corresponding templates *dropped* (3); or alternatively, templates may be dropped because superseded by more recent ones. Templates can be *merged* (4) or *split* (5), depending on the granularity and timing at which provenance needs to be created. Finally, templates may be *modified* (6) in various ways that we now discuss. Modifications may preserve the graph topology (6.1–6.4) or may alter it (6.5–6.10). Topology-preserving modifications include changing constants, changing ontology terms, and adding or dropping attributes constants. Topology-altering modifications include adding and dropping nodes and relations, and adding and dropping variables. We note that template operations 6.9-6.10 are not typically performed in isolation, but are occurring in conjunction with other changes. In practice, a template modification usually involves multiple of the changes described in Table 5.

Table 5 also shows how bindings remain correct (✓) even in the presence of modifications to templates. When a template is added, templates are merged, or a new variable is

11. In that version of smart, the expert was involved in programming a library, whose aim was to assemble the bindings' serial representation (Section 6.1.2), which was posted to a remote service, expanding them and persisting the expanded provenance.

TABLE 5
Types of Template Evolution

	Abstract Bindings	Concrete Bindings	Tabular Data	Bindings Fragment	Template compilation
1. rename template	I/S	I/S	I/S	I/S	C/R
2. add template	I	I	I	I	I
3. drop template	S	S	S	S	C
4. merge templates	I	I	I	I	C
5. split template	S	S	S	S	C
6. modify template					
6.1. change constant	✓	✓	✓	✓	✓
6.2. change ontology term	✓	✓	✓	✓	✓
6.3. add attribute	✓	✓	✓	✓	✓
6.4. drop attribute	✓	✓	✓	✓	✓
6.5. add node	✓	✓	✓	✓	✓
6.6. add relation	✓	✓	✓	✓	✓
6.7. drop node	✓	✓	✓	✓	✓
6.8. drop relation	✓	✓	✓	✓	✓
6.9. add variable	I	I	I	I	R
6.10. drop variable	S	S	S	S	C

TABLE 6
Provenance templates in smart and their changes. The column marked (N) shows the number of versions for each template. Changes are described according to the classification of Table 5.

Template Name	N	Comments
UI		
Login	1	Simple template modelling involving three PROV elements
Change_Page	1	
Use_Response	1	
Send_Request	3	Changes were prompted by iterative changes to the application's vocabulary (6.2).
Orchestrator		
Receive_Request	3	Semantic changes and vocabulary changes (6.2).
Composition	5	Revisions reflect the changes in the data structures used by the Orchestrator. The semantics of the relationships and the vocabulary were altered (6.2, 6.6).
Negotiation_Type_1	6	
Negotiation_Type_2	5	
Negotiation_Type_3	7	
Reputation Manager		
Gen_Reputation	2	The second version results from the merge with the second version of Receive_Feedback template (4).
Receive_API_Call	4	Fixed typos in vocabulary, and change of project's vocabulary (6.2).
Receive_Feedback	2	The revision reflects a change of projects's vocabulary; after the merge with Gen_Reputation, this template was abandonned in the final application (6.2, 4, 3).

added, bindings become potentially incomplete (I), resulting in a partially constructed provenance. When a template or a variable is dropped, or when templates are split, some bindings may include associations for variables that become superfluous (S), but are ignored by template expansion. Automatic bindings bean generation (Section 6.2) allows for a number of those changes to be detected at compile-time (C): a compilation error indicates that the application attempts to create bindings using incorrect names for variables or templates. In some cases, further checks can be performed on bindings at runtime (R), ensuring for instance that all variables have been bound with the required number of values.

In the smart application, there were several iterations of the templates because of the application's iterative design and distributed development. Table 6 describes how templates in smart were refined. Templates were refined by the PROV expert, but critically, consisted of changes that did not require the bindings submitted by a component to be altered, and the captured data was still valid; in other words, the changes to the template did not lead to extra development effort in the components and to data conversions. The Orchestrator required the largest number of versions because its provenance was the most complex, with respect to the number of terms in templates, and because of evolving requirements around the targeted provenance use cases. The other components required fewer changes, essentially thanks to their simplicity.

The revisions presented in Table 6 are in fact among the simplest cases of template evolution listed in Table 5. For instance, some of the template changes are due to an adjustment of the project's vocabulary. The technique we presented here complements ontology-oriented software engineering [40], by ensuring that correct URIs are included in programs to refer to the correct Semantic Web concepts. We acknowledge that the maintenance effort was particularly minimal in smart because, even though the application was evolving, its broad architecture remained stable, and the use cases for which provenance was captured did not evolve. Thus, bindings that were logged remained correct over the application development cycle.

We describe a further situation to illustrate how PROV-TEMPLATE helps software maintenance. The application picaso underwent a complete change in its templates when a type defined in an ontology had to be replaced by another type belonging to another ontology. Specifically, every occurrence of `xsd:QName` was replaced by `prov:QUALIFIED_NAME` across all templates. The reason for this change was to ensure better inter-operability with the PROV specifications (see Section 2.4). The application code was left unchanged. The database containing the stored bindings did not need to be changed either. Only the revised templates were required to be expanded again with the same bindings.

7.4 Benefit 3: Runtime and Static Checks

Definitions 4 and 5 already specify how to check that a set of bindings is compatible with a template definition. These checks can be performed at expansion time, but could also be executed at binding creation time. For instance, from a template definition, one could generate code that constructs sets of bindings, while ensuring by construction that they remain compatible with the template they are meant to be used with.

URIs are used by templates to denote types in external ontologies. Our experience shows that it is fairly frequent to introduce incorrect URIs in the definition of templates. Such a problem can be addressed in part by extracting all URIs from a template definition, and check that they have been defined in a set of imported ontologies (see Figure 6). Of course, this check is purely syntactic, and can only identify URIs that have not been declared previously. Some form of semantic reasoning would be required to detect if the correct URI has been referenced in a template.

The semantics of PROV [31] associates temporal constraints with a core subset. For a set of provenance statements to be meaningful — referred to as "valid" statements — the constraints associated with that set should be satisfiable. A necessary condition for an expanded template to be valid is that the template itself is valid. However, the expansion of a valid template is not guaranteed to generate a valid provenance graph; indeed, some bindings may for instance cause a cycle of derivations to occur in the expanded provenance graph, which would render it invalid. Given that templates are in fact provenance graphs, their validity can be checked at design time using a provenance validator (such as [41]).

Reasoning could also be applied to templates to check that they satisfy some domain-specific constraints. For instance, one may want to check that the types of entities are compatible with the types of the activities that use and generate them. Such a type of reasoning, referred to as semantic validation [42], relies on ontologies being available and referred to by templates.

While automatic methods such as validation and domain-specific reasoning are powerful, the most common automatic operation we have applied to templates is checking whether they are syntactic well-formed. Furthermore, it is important not to dismiss the power of manual methods, since such methods remain practical given the relatively small size of templates, compared to the whole provenance being generated by an application. The most common visual checks that we perform are: detecting whether a template contains disjoint graphs, detecting the presence of loops, or detecting the absence of an edge or some attribute. For instance, in Figure 1, we may want to decide that we need to provide a type attribute for the generated entity; likewise, it would have been very easy to point out that an edge is missing, should it have been the case. These tasks would have been more challenging if they had to be performed on the provenance generated by an application. In the applications of Section 2.1, we have not detected examples of invalid provenance templates, because of the continuous manual checks we performed when designing the templates.

7.5 Benefit 4: Provenance Consumption

There is a potential software engineering benefit in using PROV-TEMPLATE for applications that consume the provenance they have generated. Instead of running graph queries over provenance, applications can instead run queries over the stored bindings. This technique is used by two of our applications. The application ebook converts bindings back to workflows that can be executed, whereas picaso provides a graphical editor for the expanded templates directly from bindings.

8 Related Work

The related work is structured as follows. First, we contrast coarse and fine-grained provenance (Section 8.1); second, we survey techniques to capture provenance (Section 8.2); third, we look at an alternative to PROV-TEMPLATE and other similar graphs abstractions (Section 8.3). Finally, we position PROV-TEMPLATE in the broader context of software engineering (Section 8.4).

8.1 Coarse-Grained and Fine-Grained Provenance

Some authors [43] distinguish coarse-grained and fine-grained provenance, also commonly referred to as workflow and database provenance, respectively. The context and underpinning assumptions under which these approaches are conceived differ. In a database context, provenance explains which tables, rows, cells may have affected a query result, given a specific query that was run (for a survey of the field, we refer the reader to Cheney *et al.* [44]). Workflow provenance is regarded as more coarse-grained, because the workflow steps may not necessarily be detailed (e.g., a call to a Fast Fourier Transform) or workflow steps generate files, without the provenance of their contents being detailed.

The PROV data model is designed to express provenance and exchange it in an interoperable manner. It can be used not only to describe the flow of information, either in workflows or in database systems, but also to describe human participation in activities. When it comes to fine-grained provenance, representing it using PROV is possible, although it is unlikely to result in a compact representation that the kind of dedicated database techniques can afford [44].

8.2 Provenance: Instrumentation, Logging, Reconstructing, Compacting

A simple approach to provenance generation is to instrument code, which requires interleaving provenance-generating code in the source code. This operation is not only labor intensive, since it requires fine-tuning of provenance capturing [45] to maintain adequate performance, but it also does require both application and provenance expertise. Cross-cutting concerns of provenance generation can be addressed by aspect-oriented programming, which allows monitoring probes to be weaved into an application [20]. Instrumenting applications and generating provenance at the same time can be cumbersome from a software engineering perspective. Instead, traditional logging techniques have been combined with provenance reconstruction, in various contexts, with well-known logging tools [19], system call tracing [16], or even at the level of the operating system kernel, such as PASS [17].

Workflow systems are a class of applications generating provenance, which use a mix of instrumentation and logging (cf. Taverna [1], Vistrails [2], Kepler [3]). These systems are essentially monolithic "integrated development environments" that allow users to compose workflows and

execute them while keeping a trace of execution in the form of a provenance log.

Emerging approaches move away from such "integrated development environments" allowing disparate tools to be exploited by users. YesWorkflow [46], [47] allows scientists to annotate their scripts (in Python, R, or Perl) with special comments that reveal the main computational blocks and data flow dependencies, allowing the provenance of scientific results to be constructed and queried. To avoid code instrumentation, YesWorkflow assumes that critical information has been encoded in data product files and directory names, which allows full provenance to be inferred. Such "provenance-friendly" data organization involves URI templates, capturing how inputs and outputs are named and organized. Annotations specify notions of blocks, ports and channels, describing the static topology of the workflow. This approach is completely complementary with the one described in this article. The variables identified in the URI patterns of YesWorkflow can be used to create bindings, used by PROV-TEMPLATE to generate PROV documents.

In some situations, in particular with legacy applications, we do not have the opportunity to modify the source code in order to insert provenance-generating code, or we do not have the possibility of weaving new aspects because we cannot recompile code, or we do not have the possibility of dynamically re-loading and re-linking legacy code. In those cases, data created by such legacy applications can be mined, using application knowledge, with a view of *reconstructing* provenance [23], [24]. These approaches use a range of techniques to reconstruct provenance. Structural information allows finer grained provenance to be reconstructed, whereas content similarity allows for high-level "information flows" to be described. These approaches typically introduce some uncertainty to indicate the level of confidence associated with the reconstructed provenance relations. Again, the topology of the provenance being reconstructed can also be expressed with templates, to be instantiated with runtime values.

Finally, ProvGen is a graph generation technique for provenance [48], which relies on a seed graph, very similar to a template, combined with a set of constraints describing how it can be repeated. We conjecture that the provenance generation component could be completely decoupled from the node generation part, which would then allow PROV-TEMPLATE to be used here too.

As PROV-TEMPLATE bindings are devoid of topology information, bindings have been shown to be more compact than expanded provenance (see Section 5). There has been work investigating techniques to compact provenance, while still maintaining its queryability: Chapman *et al.* [49] propose factorization techniques, allowing common patterns of provenance information to be identified, and the amount of required storage to be reduced, while still be efficiently queryable. Our experience is that templates tend to specify the shape of provenance for small subgraphs, typically in the close vicinity of an activity. By operating over a whole provenance graph, factorization techniques stand a better chance of compressing provenance. On the other hand, the localized nature of template allows clients, submitting provenance to a provenance repository, to benefit from bindings compact representation.

8.3 Provenance Templates and Views

Curcin *et al.* [50] also propose a notion of template, seen as "a higher-level abstraction of the provenance graph data". Their templates specify "basic conceptual units that can be recorded in a provenance repository." Like PROV-TEMPLATE, their patterns use a provenance graphical model, where nodes denote concepts rather than instances, but is extended with further constructs to model sub-graph repetition. A key differentiator between their approach and PROV-TEMPLATE is that the latter has well-defined interfaces, in terms of templates and bindings in input, and expanded graphs in output, with clear standardized formats, and with a well-defined expansion algorithm.

The template language of PROV-TEMPLATE has some similarity with the provenance type graphs of Danger *et al.* [51], and Moreau's provenance summaries [52]. Provenance type graphs are combined with graph transformation techniques, such as removing and inserting nodes, to produce views over provenance graphs that satisfy some access control properties [51]. Alternatively, "user views", defined as a partition of tasks in a workflow specification [53], provide the means to selectively identify what aspect of a provenance trace should be exposed to users. PROV-TEMPLATE is intended for generating provenance, whereas provenance type graphs, summaries, and views are intended to abstract away from the concrete details of provenance. The provenance type graphs [51] build on Sun's Typed Provenance Model [54] by allowing domain specific types to be exposed. In PROV-TEMPLATE, such domain specific types can be found in the form of the `prov:type` attribute supported by the PROV data model [6]. The graph transformation technique [51] is capable of replacing subgraphs by new nodes, i.e., creating graph abstractions. This is an example of graph rewriting applied to provenance; in contrast, PROV-TEMPLATE does not perform full graph rewriting, but instead allows template nodes to be instantiated with one or more instances.

Prospective provenance, a term coined by Wilde [55], denotes the "recipe" or procedure used to compute data products. So, prospective provenance is a description of what execution is intended to be; this should be contrasted to PROV-TEMPLATE, which is a description of what provenance is to be. PROV introduces the notion of `prov:Plan`, a plan intended by an agent to achieve some goals in the context of this activity, but does not provide any details about the nature of such plans. P-Plan [56] is an approach to prospective provenance, which uses some PROV building blocks (such as activity, entity, usage, generation) to describe what an execution is intended to be like. While some relations are provided to link up actual activities and entities found in the provenance to their counterpart in the prospective provenance, P-Plan does not prescribe how provenance is to be shaped. P-Plan, like PROV-TEMPLATE, uses a notion of variable to denote actual runtime entities. ProvONE [57] is a recent community-based extension of PROV to support scientific workflow provenance. Like P-Plan, it includes prospective provenance, but replaces variables by the notion of port commonly found in workflow systems [1], [3]. The Workflow Description Ontology, which describes workflow specifications included in Research Objects [58], instead uses

a notion of datalink to specify data dependencies between the processes of a workflow.

8.4 Software Engineering and Meta Models

PRIME [39], the PRovenance Incorporating MEthodology, is a software engineering methodology to "provenance-enable" applications. It consists of three phases to be applied iteratively. First, provenance use cases need to be elicited to identify the type of functionality that is expected out of provenance information. Second, the application is deconstructed into actors, processes, and information flows. Third, information items are exposed, provenance is then captured, and provenance functionality is implemented. PRIME does not specify how provenance is generated. Again, PROV-TEMPLATE has a natural place in this software engineering methodology, since the templates can be expressed by the designer to address some provenance use cases, and can directly be used for provenance generation. More recently, a set of common provenance *recipes* [14] has been put forward for provenance patterns commonly encountered in applications. Likewise, ProvErr [59] relies on engineers' application knowledge to construct a *dependency model* aimed to support root cause analysis of system faults. Both provenance recipes and dependency models are good candidates for creating provenance templates.

Zhu and Bayley [60] propose an algebra of design patterns, from which they derive a notion of equivalence between pattern expressions and a normal form for pattern expressions. It is an open question as to whether a similar algebra can be developed for provenance templates, allowing some reasoning to be made about how templates are composed.

PROV provenance draws upon the linked data [61] and Semantic Web approaches [62]. In this context, significant attention has been given to the problem of bringing knowledge and software engineering together [40]. In particular, there has been a growing interest in applying ontologies to the various stages of the software engineering life cycle (for an overview, see [40]). For instance, there are approaches allowing conversion between OWL ontologies and UML models, and vice-versa [63]. As we have already shown in Figure 6, PROV-TEMPLATE builds on ontologies by making explicit references to classes and properties defined in ontologies. This presents researchers with opportunities to integrate PROV-TEMPLATE in the software engineering process to bring forth its benefits (as discussed in Section 7).

9 Conclusion and Future Work

Ease of generation remains an adoption hurdle for provenance technology. To address this challenge, we have presented PROV-TEMPLATE, a practical approach that facilitates the generation of provenance. It consists of three parts. Templates provide a declarative way of specifying the provenance to be generated, with placeholders (referred to as variables) for values to be filled. Sets of bindings are simple JSON data structures associating variable names to values. An expansion algorithm creates a provenance document from a template, by replacing all variables by values found in a set of bindings. The expansion algorithm is capable of dealing with multiple values for variables. The approach is implemented by the ProvToolbox library, an open source library for manipulating provenance in Java.

Our quantitative evaluation shows that exchanging sets of bindings rather than provenance documents incurs a significantly reduced cost in communications and/or storage, as the size of bindings is demonstrated to be on average 40% of that of expanded provenance documents. The performance evaluation also shows that the approach is tractable, with only fractions of milliseconds required for expanding typical templates.

Our practical experience with PROV-TEMPLATE over the course of two years has shown four benefits provided by the approach. It helps with separation of responsibilities, allowing distributed developers to focus on code writing and information logging, whereas a PROV expert can focus on the design of provenance templates and their deployment in an application. PROV-TEMPLATE facilitates maintenance of provenance since it allows minor revisions of provenance to be supported, without having to modify the application, as long as the templates still rely on the same logged values. PROV-TEMPLATE allows for an application-wide library of templates to be assembled, and a series of static and dynamic checks to be supported; these checks help the application log the necessary information to create provenance correctly. Finally, PROV-TEMPLATE allows for applications that consume their own provenance to exploit the regular structure of bindings, rather than having to rely on graph queries over provenance.

There are a number of opportunities to build upon PROV-TEMPLATE in future work. If a designer specifies all the provenance to be generated in an application by means of templates, there is only a need to store sets of bindings and templates. Thus, we could envisage a notion of "provenance repository" [64], in which PROV-compatible provenance is only generated on demand, and is not persisted in that form. Instead, the only information that needs to be captured and stored is templates and sets of bindings. Pushing this approach to its logical end, the idea of a provenance template management system becomes crucial, with key functionality, including editing and storage of templates and their versions, migration of bindings to new templates, and on-the-fly PROV-compatible provenance generation. Templates can further be "compiled" into code that generates PROV efficiently from a set of bindings. Services for posting bindings can be generated automatically from templates, and perform compatibility checks directly, giving early feedback to developers if something goes wrong.

Provenance is typically queried by means of graph queries, expressed in languages such as SPARQL. With the PROV-TEMPLATE approach, provenance now consists of templates and bindings. Thus, provenance graph queries can be optimized by developing query plans that take into account the static nature of provenance templates, and by directly querying the bindings, which can also be indexed to improve performance.

A theoretical strand of work could investigate the meaning of abstract graphs such as templates. There, nodes no longer represent instances but sets of these. With such a notion, one can also study the set of algebraic operations to process such graphs and the type of reasoning that is

possible over such abstract graphs.

ACKNOWLEDGMENTS

This work is funded in part by the EPSRC SOCIAM (EP/J017728/1) and ORCHID (EP/I011587/1) projects, the FP7 SmartSociety (600854) project, and the ESRC eBook (ES/K007246/1) project.

The data referred to as smart, ebook, and picaso supporting this study are openly available from the University of Southampton repository at DOI: 10.5258/SOTON/390436. As far as food data are concerned, their sets of bindings cannot be made openly available because they contain commercially sensitive data, but the templates and measurements are available.

REFERENCES

[1] P. Alper, K. Belhajjame, C. A. Goble, and P. Karagoz, "Enhancing and abstracting scientific workflow provenance for data publishing," in *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. New York, NY, USA: ACM, 2013, pp. 313–318. [Online]. Available: http://doi.acm.org/10.1145/2457317.2457370

[2] C. Silva, E. Anderson, E. Santos, and J. Freire, "Using vistrails and provenance for teaching scientific visualization," *Computer Graphics Forum*, vol. 30, no. 1, pp. 75–84, 3 2011. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2010.01830.x

[3] I. Altintas, O. Barney, and E. Jaeger-Frank, "Provenance collection support in the kepler scientific workflow system," in *Proceedings of the 2006 International Conference on Provenance and Annotation of Data (IPAW'06)*. Springer, 2006, pp. 118–132. [Online]. Available: http://dx.doi.org/10.1007/11890850_14

[4] F. Chirigati, D. Shasha, and J. Freire, "Reprozip: Using provenance to support computational reproducibility," in *Proceedings of the 5th USENIX Conference on Theory and Practice of Provenance (TaPP'13)*. Berkeley, CA, USA: USENIX Association, 2013, pp. 1–1. [Online]. Available: http://dl.acm.org/citation.cfm?id=2482613.2482614

[5] S. Ramchurn, E. Simpson, J. Fischer, T. D. Huynh, Y. Ikuno, S. Reece, W. Jiang, F. Wud, J. Flann, S. J. Roberts, L. Moreau, T. Rodden, and N. Jennings, "HAC-ER: A disaster response system based on human-agent collectives," Istambul, Turkey, May 2015. [Online]. Available: http://eprints.soton.ac.uk/374070/

[6] L. Moreau, P. Missier (eds.), K. Belhajjame, R. B'Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, T. Lebo, J. McCusker, S. Miles, J. Myers, S. Sahoo, and C. Tilmes, "PROV-DM: The PROV Data Model," World Wide Web Consortium, W3C Recommendation REC-prov-dm-20130430, Oct. 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-prov-dm-20130430/

[7] R. Bose and J. Frew, "Lineage retrieval for scientific data processing: A survey," *ACM Computing Surveys*, vol. 37, no. 1, pp. 1–28, Mar. 2005. [Online]. Available: http://homepages.inf.ed.ac.uk/rbose/pubs/bose_2005_ACM_CS.pdf

[8] S. Miles, P. Groth, M. Branco, and L. Moreau, "The requirements of recording and using provenance in e-science experiments," *Journal of Grid Computing*, vol. 5, no. 1, pp. 1–25, 2007. [Online]. Available: http://eprints.ecs.soton.ac.uk/10269/

[9] L. Moreau, "The foundations for provenance on the Web," *Foundations and Trends in Web Science*, vol. 2, no. 2–3, pp. 99–241, Nov. 2010. [Online]. Available: http://eprints.ecs.soton.ac.uk/21691/

[10] Y. Gil, J. Cheney, P. Groth, O. Hartig, S. Miles, L. Moreau, and P. Pinheiro da Silva, "Provenance XG final report," World Wide Web Consortium, Tech. Rep., 2010. [Online]. Available: http://www.w3.org/2005/Incubator/prov/XGR-prov-20101214/

[11] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman, "Information accountability," *Commun. ACM*, vol. 51, no. 6, pp. 81–87, Jun. 2008. [Online]. Available: http://hdl.handle.net/1721.1/37600

[12] L. Moreau, "Provenance-based reproducibility in the semantic web," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 9, pp. 202–221, Feb. 2011. [Online]. Available: http://eprints.ecs.soton.ac.uk/21992/

[13] T. D. Huynh, M. Ebden, M. Venanzi, S. Ramchurn, S. Roberts, and L. Moreau, "Interpretation of crowdsourced activities using provenance network analysis," in *Conference on Human Computation and Crowdsourcing (HCOMP'13)*, Nov. 2013. [Online]. Available: http://www.aaai.org/ocs/index.php/HCOMP/HCOMP13/paper/view/7388

[14] L. Moreau and P. Groth, *Provenance: An Introduction to PROV*. Morgan and Claypool, September 2013.

[15] F. Curbera, Y. Doganata, A. Martens, N. K. Mukhi, and A. Slominski, "Business provenance – a technology to increase traceability of end-to-end operations," in *OTM 2008 Confederated International Conferences*. Springer, 2008, pp. 100–119. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88871-0_10

[16] J. Frew, D. Metzger, and P. Slaughter, "Automatic capture and reconstruction of computational provenance," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 485–496, 2008.

[17] D. A. Holland, M. Seltzer, U. Braun, and K.-K. Muniswamy-Reddy, "Pass-ing the provenance challenge," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, 2008. [Online]. Available: http://www3.interscience.wiley.com/journal/116316566/abstract

[18] L. Moreau and P. Groth, "Provenance of publications: A PROV style for latex," in *Seventh USENIX Workshop on the Theory and Practice of Provenance (TAPP'15)*. Edinburgh, Scotland: USENIX, Jul. 2015. [Online]. Available: http://eprints.soton.ac.uk/378019/

[19] D. Ghoshal and B. Plale, "Provenance from log files: A bigdata problem," in *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, ser. EDBT '13. New York, NY, USA: ACM, 2013, pp. 290–297. [Online]. Available: http://doi.acm.org/10.1145/2457317.2457366

[20] P. Brauer, F. Fittkau, and W. Hasselbring, "The aspect-oriented architecture of the CAPS framework for capturing, analyzing and archiving provenance data," in *Provenance and Annotation of Data and Processes*, vol. 8628. Springer, 2015, pp. 223–225.

[21] J. Cheney, A. Ahmed, and U. A. Acar, "Provenance as dependency analysis," *Mathematical Structures in Computer Science*, vol. 21, no. 6, pp. 1301–1337, 2011. [Online]. Available: http://dx.doi.org/10.1017/S0960129511000211

[22] J. Cheney, "Program slicing and data provenance," *IEEE Data Engineering Bulletin*, pp. 22–28, December 2007.

[23] S. Magliacane, "Reconstructing provenance," in *The Semantic Web — ISWC 2012*, vol. 7650. Springer, 2012, pp. 399–406. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35173-0_29

[24] T. De Nies, I. Taxidou, A. Dimou, R. Verborgh, P. M. Fischer, E. Mannens, and R. Van de Walle, "Towards multi-level provenance reconstruction of information diffusion on social media," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM '15)*. New York, NY, USA: ACM, 2015, pp. 1823–1826. [Online]. Available: https://websci.informatik.uni-freiburg.de/publications/cikm2015-multilevel-provenance

[25] T. Lebo, S. Sahoo, D. McGuinness (eds.), K. Behajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, and J. Zhao, "PROV-O: The PROV Ontology," World Wide Web Consortium, W3C Recommendation REC-prov-o-20130430, Oct. 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-prov-o-20130430/

[26] L. Moreau, P. Missier (eds.), J. Cheney, and S. Soiland-Reyes, "PROV-N: The Provenance Notation," World Wide Web Consortium, W3C Recommendation REC-prov-n-20130430, Oct. 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-prov-n-20130430/

[27] H. Hua, C. Tilmes, S. Zednik (eds.), and L. Moreau, "PROV-XML: The PROV XML Schema," World Wide Web Consortium, W3C Working Group Note NOTE-prov-xml-20130430, Apr. 2013. [Online]. Available: http://www.w3.org/TR/2013/NOTE-prov-xml-20130430/

[28] [Online]. Available: https://pypi.python.org/pypi/prov

[29] L. Moreau, "ProvToolbox — Java library to create and convert W3C prov data model representations," http://lucmoreau.github.io/ProvToolbox/, Apr. 2016.

[30] T. D. Huynh, P. Groth, and S. Zednik (eds.), "PROV implementation report," World Wide Web Consortium, W3C Working Group Note NOTE-prov-overview-20130430, April 2013. [Online]. Available: http://www.w3.org/TR/2013/NOTE-prov-implementations-20130430/

[31] J. Cheney, P. Missier, L. Moreau (eds.), and T. D. Nies, "Constraints of the PROV Data Model," World

Wide Web Consortium, W3C Recommendation REC-prov-constraints-20130430, Oct. 2013. [Online]. Available: http://www.w3.org/TR/2013/REC-prov-constraints-20130430/

[32] D. Michaelides, T. D. Huynh, and L. Moreau, "PROV-TEMPLATE: A template system for prov documents," Jun. 2014, technical Note. [Online]. Available: https://provenance.ecs.soton.ac.uk/prov-template-2014-06-07/

[33] H. S. Packer, L. Dragan, and L. Moreau, "An auditable reputation service for collective adaptive systems," in *Social Collective Intelligence: Combining the Powers of Humans and Machines to Build a Smarter Society*, D. Miorandi, V. Maltese, M. Rovatsos, A. Nijholt, and J. Stewart, Eds. Springer, August 2014, pp. 159–184. [Online]. Available: http://eprints.soton.ac.uk/365559/

[34] D. Michaelides, R. Parker, C. Charlton, W. Browne, and L. Moreau, "Intermediate notation for provenance and workflow reproducibility," in *6th International Provenance and Annotation Workshop (IPAW'16)*, McLean, VA, US, Jun. 2016, pp. 1–12. [Online]. Available: http://eprints.soton.ac.uk/393117/

[35] L. Moreau, B. Ludaescher, I. Altintas, R. S. Barga, S. Bowers, S. Callahan, G. Chin Jr., B. Clifford, S. Cohen, S. Cohen-Boulakia, S. Davidson, E. Deelman, L. Digiampietri, I. Foster, J. Freire, J. Frew, J. Futrelle, T. Gibson, Y. Gil, C. Goble, J. Golbeck, P. Groth, D. A. Holland, S. Jiang, J. Kim, D. Koop, A. Krenek, T. McPhillips, G. Mehta, S. Miles, D. Metzger, S. Munroe, J. Myers, B. Plale, N. Podhorszki, V. Ratnakar, E. Santos, C. Scheidegger, K. Schuchardt, M. Seltzer, Y. L. Simmhan, C. Silva, P. Slaughter, E. Stephan, R. Stevens, D. Turi, H. Vo, M. Wilde, J. Zhao, and Y. Zhao, "The First Provenance Challenge," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 409–418, Apr. 2008. [Online]. Available: http://www.ecs.soton.ac.uk/~lavm/papers/challenge-editorial.pdf

[36] P. V. Biron and A. Malhotra, "XML Schema Part 2: Datatypes," Oct. 2004. [Online]. Available: http://www.w3.org/TR/xmlschema-2/

[37] G. Carothers and E. Prud'hommeaux, "RDF 1.1 Turtle," World Wide Web Consortium, W3C Recommendation, Feb. 2014. [Online]. Available: http://www.w3.org/TR/2014/REC-turtle-20140225/

[38] J. Tennison and Gregg Kellogg (eds.), "Model for tabular data and metadata on the web," World Wide Web Consortium, Recommendation, 2015. [Online]. Available: https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/

[39] S. Miles, P. Groth, S. Munroe, and L. Moreau, "Prime: A methodology for developing provenance-aware applications," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, pp. 1–42, August 2011. [Online]. Available: http://eprints.ecs.soton.ac.uk/17450/

[40] H.-J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *Proc. of Workshop on Sematic Web Enabled Software Engineering"(SWESE) on the ISWC*, 2006, pp. 5–9. [Online]. Available: https://km.aifb.kit.edu/ws/swese2006/final/happel_full.pdf

[41] L. Moreau, T. D. Huynh, and D. Michaelides, "An online validator for provenance: Algorithmic design, testing, and api," in *17th International Conference on Fundamental Approaches to Software Engineering (FASE'14)*, vol. 8411. Springer, April 2014, pp. 291–305. [Online]. Available: http://eprints.soton.ac.uk/361113/

[42] S. Miles, S. C. Wong, W. Fang, P. Groth, K.-P. Zauner, and L. Moreau, "Provenance-based validation of e-science experiments," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 1, pp. 28–38, 2007. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2006.11.003

[43] W.-C. Tan, "Provenance in databases: Past, current, and future," *Bulletin of the Technical Committee on Data Engineering*, vol. 30, no. 4, pp. 3–12, Dec. 2007. [Online]. Available: ftp://ftp.research.microsoft.com/pub/debull/A07dec/wang-chiew.pdf

[44] J. Cheney, L. Chiticariu, and W.-C. Tan, "Provenance in databases: Why, how, and where," *Found. Trends databases*, vol. 1, no. 4, pp. 379–474, Apr. 2009. [Online]. Available: http://dx.doi.org/10.1561/1900000006

[45] P. Groth and L. Moreau, "Recording process documentation for provenance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 9, pp. 1246–1259, Sep. 2009. [Online]. Available: http://www.ecs.soton.ac.uk/~lavm/papers/tpds09.pdf

[46] T. M. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, K. Bocinsky, Y. Cao, F. Chirigati, S. C. Dey, J. Freire, D. N. Huntzinger, C. Jones, D. Koop, P. Missier, M. Schildhauer, C. R. Schwalm, Y. Wei, J. Cheney, M. Bieda, and B. Ludäscher, "Yesworkflow: A user-oriented, language-independent tool for recovering workflow information from scripts," *International Journal of Digital Curation*, vol. 10, no. 1, 2015. [Online]. Available: http://arxiv.org/abs/1502.02403

[47] T. McPhillips, S. Bowers, K. Belhajjame, and B. Ludäscher, "Retrospective provenance without a runtime provenance recorder," in *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)*. Edinburgh, Scotland: USENIX Association, Jul. 2015. [Online]. Available: https://www.usenix.org/conference/tapp15/workshop-program/presentation/mcphillips

[48] H. Firth and P. Missier, "Provgen: Generating synthetic prov graphs with predictable structure," in *Provenance and Annotation of Data and Processes*, B. Ludäscher and B. Plale, Eds., vol. 8628. Springer, 2015, pp. 16–27. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-16462-5_2

[49] A. P. Chapman, H. V. Jagadish, and P. Ramanan, "Efficient provenance storage," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, 2008, pp. 993–1006. [Online]. Available: http://doi.acm.org/10.1145/1376616.1376715

[50] V. Curcin, E. Fairweather, R. Danger, and D. Corrigan, "Templates as a method for implementing data provenance in decision support systems," *Journal of Biomedical Informatics*, vol. 65, pp. 1–21, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.jbi.2016.10.022

[51] R. Danger, V. Curcin, P. Missier, and J. Bryans, "Access control and view generation for provenance graphs," *Future Generation Computer Systems*, vol. 49, pp. 8 – 27, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X1500031X

[52] L. Moreau, "Aggregation by provenance types: A technique for summarising provenance graphs," in *Graphs as Models 2015 (An ETAPS'15 workshop)*, London, UK, Apr. 2015, pp. 129–144. [Online]. Available: http://eprints.soton.ac.uk/364726/

[53] S. Cohen-Boulakia, O. Biton, S. Cohen, and S. Davidson, "Addressing the provenance challenge using zoom," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 5, pp. 497–506, 2008. [Online]. Available: http://dx.doi.org/10.1002/cpe.1232

[54] L. Sun, J. Park, and R. Sandhu, "Engineering access control policies for provenance-aware systems," in *Proceedings of the Third ACM Conference on Data and Application Security and Privacy (CODASPY '13)*. New York, NY, USA: ACM, 2013, pp. 285–292. [Online]. Available: http://dx.doi.org/10.1145/2435349.2435390

[55] B. Clifford, I. Foster, J.-S. Voeckler, M. Wilde, and Y. Zhao, "Tracking provenance in a virtual data grid," *Concurr. Comput. : Pract. Exper.*, vol. 20, no. 5, pp. 565–575, Apr. 2008. [Online]. Available: http://dx.doi.org/10.1002/cpe.v20:5

[56] D. Garijo and Y. Gil, "Augmenting PROV with Plans in P-PLAN: Scientific Processes as Linked Data," in *Second International Workshop on Linked Science: Tackling Big Data (LISC), held in conjunction with the International Semantic Web Conference (ISWC)*, Boston, MA, 2012. [Online]. Available: http://www.isi.edu/~gil/papers/garijo-gil-lisc12.pdf

[57] V. Cuevas-Vicenttin, B. Ludaescher, P. Missier, K. Belhajjame, F. Chirigati, Y. Wei, S. Dey, P. Kianmajd, D. Koop, S. Bowers, and I. Altintas, "Provone: A prov extension data model for scientific workflow provenance," DataOne Project, Tech. Rep., Mar. 2014. [Online]. Available: http://vcvcomputing.com/provone/provone.html

[58] K. Belhajjame, J. Zhao, D. Garijo, M. Gamble, K. Hettne, R. Palma, E. Mina, O. Corcho, J. M. Gómez-Pérez, S. Bechhofer, G. Klyne, and C. Goble, "Using a suite of ontologies for preserving workflow-centric research objects," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 32, pp. 16 – 42, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570826815000049

[59] P. Chen and B. A. Plale, "Proverr: System level statistical fault diagnosis using dependency model," in *15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2015, Shenzhen, China, May 4-7, 2015*, 2015, pp. 525–534. [Online]. Available: http://dx.doi.org/10.1109/CCGrid.2015.86

[60] H. Zhu and I. Bayley, "An algebra of design patterns," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 3, pp. 23:1–23:35, Jul. 2013. [Online]. Available: http://doi.acm.org/10.1145/2491509.2491517

[61] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, 1st ed. Morgan & Claypool, 2011. [Online]. Available: http://linkeddatabook.com/

[62] N. Shadbolt, T. Berners-Lee, and W. Hall, "The semantic web revisited," *IEEE Intelligent Systems*, vol. 21, no. 3, pp. 96–101, May 2006. [Online]. Available: http://dx.doi.org/10.1109/MIS.2006.62
[63] S. Brockmans, R. M. Colomb, P. Haase, E. F. Kendall, E. K. Wallace, C. Welty, and G. T. Xie, "The 5th international semantic web conference (iswc'06)." Springer, 2006, pp. 187–200. [Online]. Available: http://dx.doi.org/10.1007/11926078_14
[64] T. D. Huynh and L. Moreau, "ProvStore: a public provenance repository," in *5th International Provenance and Annotation Workshop (IPAW'14)*, June 2014, pp. 275–277. [Online]. Available: http://eprints.soton.ac.uk/365509/

Luc Moreau is Professor of Computer Science and head of the Web and Internet Science group, in the Department of Electronics and Computer Science, at the University of Southampton. Luc is a leading figure in the area of data provenance. He was co-chair of the W3C Provenance Working Group that produced the PROV recommendations. He is co-investigator of the ORCHID, SOCIAM, SmartSociety, and eBook projects.

Belfrit Victor Batlajery is a PhD student in the Web and Internet Science group, in the Department of Electronics and Computer Science at the University of Southampton. He completed his Master Degree in Business Informatics and has interests in Provenance and Semantic Web. His undergoing PhD project is about Food Provenance where he investigates the role of provenance to support due diligence in the food industry.

T Dong Huynh is a researcher in the Web and Internet Science group, in the Department of Electronics and Computer Science, at the University of Southampton. He has extensive experience in the areas of trust, reputation and provenance. Dong pioneered the provenance network analytics method to classify data based on their provenance. He led the development of PICASO, CollabMap, and ProvStore and is the main author of PROV-JSON and the PROV Python package.

Danius Michaelides is a researcher in the Web and Internet Science group. His research interests include eScience, distributed computing, distributed information management and Semantic Web technologies. He is currently working on a project funded by the UK's Economic and Social Research Council to build novel tools for training and research in quantitative social science, including the EBook application.

Heather Packer is a Research Fellow in the Web and Internet Science group, in the Department of Electronics and Computer Science at the University of Southampton, where she also obtained her doctoral degree in Computer Science for algorithms for the automatic generation of lightweight ontologies for task focused domains. In the SmartSociety project, she is focusing on provenance, reputation, transparency and accountability, in particular in Smartshare.